Decontamination from Black Viruses Using Parallel Strategies

by

Yichao Lin

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the Master degree in
Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

In this thesis, we consider the problem of decontaminating networks from *black viruses* (BVs) with a team of mobile agents, using parallel strategies. The BV is a harmful process whose initial location is unknown a priori. It destroys any agent arriving at the network site where it resides and, once triggered, it spreads to all the neighboring sites, creating copies of itself, thus increasing its presence in the network. To eliminate a virus present in a node, an agent has to move on that node; however, once the disinfection is performed, the agent is destroyed (i.e., it becomes inactive and cannot operate anymore). Existing literature has proposed sequential strategies that minimize the spread of the virus, such techniques are however quite inefficient in terms of time complexity. In order to permanently remove any presence of the BV faster, still minimizing the number of site infections, we propose parallel strategies to decontaminate the BVs. Instead of exploring the network sequentially, we employ a group of agents that coordinate to follow a collective protocol to explore the network simultaneously, thus dramatically reducing the time needed in the exploration phase but still keeping the spread (and the number of agents loss) asymptotically optimal. Different protocols are proposed in meshes, tori, and chordal rings following the monotonicity principle (i.e., once a node is disinfected we prevent it from being recontaminated). Finally, a solution is proposed also for the general case of the arbitrary topology. We analyze theoretically the cost of all our solutions for special topologies showing the advantages of our strategies with respect to the existing ones. In the case of the arbitrary topology, we conduct experimental analysis to assess the performance of our solution, confirming its efficiency. In all cases, our strategies dramatically improve time while maintaining asymptotically optimal spread and agent losses.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Professor Paola Flocchini for her enthusiastic motivation, judicious and enlightening advices, and patient guidance throughout this thesis work! Prof.Flocchini carefully selected my courses and thesis topics, and at the same time, consistently allowed this thesis to be my own work, but steered me in the right direction whenever she thought I need it. Without her continuous support and encouragement through the process of writing this thesis, it would not be possible for me to complete this work.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement. Without my family's encouragement and support, I would not achieve this.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

A distributed system is a group of computational entities cooperating with each other to achieve one or more tasks. This thesis deals with distributed computing by mobile agents in networks. More specifically, we deal with the problem of deploying a group of mobile agents that follow the same protocol, to explore the network and decontaminate it from a dangerous virus (called Black Virus) present on some of the the network nodes. In this Chapter, the motivations of the problem are provided, following is a brief summary of the contributions. Finally, an overview of the organization of the thesis is presented.

1.1 Problem and Motivation

Mobile agents are widely used in distributed systems and networks. Their employ can cause some security issues, thus threatening the network. For example, a contaminated or infected host can destroy working agents for various malicious purposes. A malicious agent can contaminate or infect other computer nodes so they become malfunctional or crashed. These static harmful hosts, often called *Black Holes* (BH) trigger the problem called *Black Hole Search* (BHS), the focus of which is to locate their positions. This problem has been studied in several variants; in particular, in different topologies, and under different assumptions on synchrony level (synchronous vs. asynchronous). Harmful agents trigger instead the problem called *Intruder Capture* (IC). The main focus of the Intruder Capture is to deploy a group of mobile agents to capture an extraneous mobile agent (the intruder) that moves arbitrarily fast through the network and infects the visiting sites. Also this problem has been investigated in a variety of topologies. A detailed literature review will be provided in Chapter 2. Note that a black hole is static and damages only the agents reaching it without leaving any detectable trace. The intruder, instead, is mobile and harmful to the network nodes, but does not cause any harm to other system agents. A new harmful presence called *black virus* BV has been introduced by Cai et al. in [5]. A BV is a dangerous process that resides at an unknown site in a network and destroys any upcoming agents, but unlike the BH, the node where the original BV resides becomes clean when an agent reaches it. At the same time, the original BV multiplies, creating clones of itself, and spreads to all neighbouring nodes, thus increasing its presence and damage in the network. A BV is destroyed when it moves to a site where there is already an agent. Based on this harmful presence, a new problem called *Black Virus Decontamination* (BVD) is presented by Cai et al., the main focus of which is to use a group of system agents to permanently remove any presence of the BV from the network. A protocol defining the actions of the agents solves the BVD problem if at least one agent survives and the network is free of BVs. Also, a desirable property of a decontamination protocol is that the nodes which have been explored or cleaned by mobile agents are not recontaminated by the BV spreading. A solution protocol with such a property is called *monotone* (see [27]). Some important cost measures are: the number of node infections by the BVs (casualties); the size of the team, i.e, the number of agents employed by the solution, and the time needed by the solution. Solutions in which the agents explore the network's nodes sequentially have been proposed in [1, 5, 6] in various topologies. The size of the team is minimizes in [5, 6] and the number of site infections is also minimized in such a case. However, in all those solutions, the time cost is quite high and not scalable, as it is linear in the size of the network. In the thesis, we are interested in faster solutions using parallel strategies, where we deploy a larger number of mobile agents following the same protocol to decontaminate the network concurrently, with the goal to decrease time. For our strategies, we also introduce a new measure, the *total working time* (TWT), which is obtained as a combination of all three classical costs (number of agents, execution time, casualties), and we compare it with the TWT of the existing algorithms.

1.2 Our Contribution

1. In this thesis, we propose parallel strategies to solve the BVD problem. This is the first attempt to deal with this issue in a parallel way. Like in previous work, agents are not allowed to communicate with each other unless they are in the same network node so the protocol should enable the agents in different nodes to move independently but in a synchronized fashion to achieve the global goal. We provide a simple efficient solution to deal with this problem. We also give the size of the minimum exploring team to guarantee both the TWT and the casualties we reach.

2. The BVD problem is investigated for three important topologies: *meshes*, *tori*, *chordal rings*. All the protocols are asymptotically optimal both in term of TWT and casualties. We compare our solutions with [1] and [5], where the exploring route is sequential, and the result is that our solution outperforms theirs in terms of both TWT and casualties. We should point out that in chordal rings especially, the more complicated the chordal ring becomes, the higher are the improvements of our algorithms in terms of in TWT compared to [1].

3. The BVD problem is investigated also in the general case of the arbitrary topology. In this case we run extensive simulations to test its performance and we experimentally observe that also in this case, our solution is better than the existing sequential one.

1.3 Thesis Organization

The thesis is organized as follows:

Chapter 2 contains a literature review on related problems. We begin by reviewing the Black Hole Search and Intruder Capture problem, we then focus on the solution to the BVD problem where the mobile agents explore the network sequentially. The problem has been studied in different topologies: two-dimensional grids, three-dimensional grids, tori, chordal rings, hypercubes and arbitrary network. Also, the variant of this problem, which is decontamination of an arbitrary network from multiple black virus, is reviewed.

Chapter 3 introduces terminology, definitions and model for the BVD problem used in the rest of the thesis. We also describe the high level ideas that serve as the basis of all our solutions. Since monontonicity is a necessary condition for spread optimality, we explain this concept and draw some conclusions.

Chapter 4 focuses on the BVD problem for the mesh topology. In this chapter, an optimal algorithm in terms of casualties and TWT is developed. Complexity analysis is performed and results obtained. Some comparisons are also made between our solution and [5] and the result shows that our solution is better.

Chapter 5 presents the BVD problem for the chordal ring topology. In this chapter, we introduce the *Three Jump Notifying Technique* (TJNT) to manipulate each mobile agent to efficiently move along its route during the exploration, and to avoid the spread of clone BVs after the original BV is triggered. Based on this technique, we develop the parallel strategy for the mobile agents to decontaminate the chordal ring. Complexity analysis in terms of casualty and TWT are performed. Finally some comparisons are made between our solution and [1] and the result shows that our solution performs better.

Chapter 6 proposes two parallel strategies to solve the BVD problem in the arbitrary graph: Flood Strategy and Castle-First Strategy. Flood Strategy completes the decontamination in $2d$ unit of time where d is the diameter of the graph with a very high cost in terms of agents compared to any other strategy; on the other hand, the CASTLE-FIRST Strategy reaches a compromise between the sequential strategy and the FLOOD Strategy: it employs a reasonable number of agents and significantly less time than the sequential strategy.

In Chapter 7, we experimentally study the BVD problem in the arbitrary graph. Details of simulation work are presented. We run experiments on different size of graphs with many connectivity densities and the results are collected and analyzed.

Chapter 8 summaries the main conclusion of our work and present some open problems and future work.

Chapter 2

Literature Review

Note floc and floc are the same paper.

Some papers might have appeared in Journal, you need to update the references.

Also note, the reference should be in alphabetic order

Some references did not compile, perhaps I don't have the last version of the bib file

Mobile agents have been widely used in distributed computing due to their features, especially mobility, which allow them to migrate between computers at any time during their execution. A group of agents can be used to perform a variety of tasks, for example, network exploration, monitoring, maintenance, etc. However, the introduction of mobile agents tend to cause security problems, possibly threatening the network. Various security issues and solution algorithms have been proposed in [26]. Generally, the threat that the mobile agents cause are divided into two categories: in first case, the malicious agents can cause network nodes malfunction or crash by contaminating or infecting them (*harmful agents*); in second case, the contaminated or infected hosts can destroy working agent for various malicious purposes (*harmful hosts*). These two threats trigger two problems: *Black Hole Search* (BHS) and *Intruder Capture* (IC), which will be introduced in the following Sections. We then review the BVD problem, which deals with the decontamination of a harmful presence that cause the network node malfunction, leaves the network node clean when it is triggered, and spreads to all its neighbouring nodes increasing its presences. In the section introducing BVD problem, we describe the solutions that have been proposed to decontaminate the networks, and review different decontamination algorithms based on different strategies.

2.1 Black Hole Search, BHS

The BHS problem assumes there is a BH (or multiple static BHs) residing at certain network nodes. The BHs will destroy any upcoming agents without leaving any detectable trace. The task is to use a team of good agents to locate the black hole(s). The task is completed when at least one agent survives and reports the location(s) of the black hole(s). Note that this task is dangerous for the team of agents and some will inevitably be destroyed during their search.

The BHS problem has been widely studied in various topologies and settings: for example, in asynchronous rings and tori [7, 8], in directed graphs [12], in arbitrary graphs [18], in anonymous rings [16], in common interconnection networks [19]. The possibility of using multiple agents has been studied in [10], locating a blacking hole without using the knowledge of incoming link has been considered in [29]. Moreover, in [14] the computational complexity of searching for a black hole is studied in [34, 35] approximation and hardness results about the black hole search are derived.

The main differences made in the literature is whether the system is synchronous or asynchronous. In the synchronous model, one agent is sent to explore a node, and other agents can know whether the node is safe or not by using timeouts: i.e., waiting until a bounded time expires. Synchronous model allows detection of unknown number of multiple BH's. In asynchronous model, there is not bounded agent's travelling time on link, so there is no way to distinguish between a slow link and black hole. The "cautious walk" is the main exploring method to check if the node is safe or not. If one agent is sent to explore a node, before the node is confirmed safe, no more agents are sent to the node. The asynchronous model only allows detection of a known number of BH's with an additional assumption that the number of nodes in the topology is known. In [18, 20, 41], the authors study a variation of model where communication among agents is achieved by placing tokens on the nodes. In [9], the case of black hole located on links instead of nodes in investigated. In [13] the case of black holes located in synchronous tree networks are studied for a given starting node. In [11], the authors consider locating and repairing faults in network (the agent would die after repairing the fault). The authors compute the number of steps that is necessary and sufficient to complete this task. Finally, some recent studies deal with scattered agents searching for a black hole in rings and toris [7, 9]. Also, in [30, 31, 38, 39] various methods to protect mobile agent against malicious host are discussed. Note that the BH is static which means that it does not propagate in the network and thus it is not harmful to other sites.

What is worth pointing out is that the number of BHs remains the same as it is at the beginning of the search, thus not causing harm to other sites of the network.

2.2 Intruder Capture

The IC problem assumes that there is an intruder moving with an arbitrary speed from node to node in the network and contaminating the sites it visits. The goal is to deploy a group of mobile agents to capture the intruder; the intruder is captured when it comes in contact with an agent. Note that the intruder does not cause any harm to the up-coming agents. This problem is equivalent to the problem of decontaminating a network contaminated by a virus while avoiding any recontamination, which is also called *connected graph search*, and has been extensively studied (see [28]). The Intruder Capture problem is first introduced in [3] and has been widely investigated in a variety network topologies:

trees [2, 3, 17], hypercubes[23], multi-dimensional grids[23], pyramids[40],chordal rings[25], tori[22], outerplanar graphs[33], etc. The studies of arbitrary graph has been started in [4, 32]. Note that monotone is a critical principle in the solutions of IC problems.

2.3 Agent Capabilities

Different capacities granted to the mobile agents have an impart on solving the BHS problem, IC problem and also the BVD problem.We discuss these capabilities in the following section.

Communication Mechanisms Mobile agents can communicate with each other only when they are in the same node in a network. Various communication methods have been studied in literature: whiteboard, tokens and time-out.

In [12, 18, 24, 36], the whiteboard model is used, which is a storage space located at each node and agents arriving there are able to read and write. In the token model, (see [8, 24]), tokens are like pebbles that the agents can drop off and pick up at nodes or edges. Time-out mechanisms achieve implicit communication (or synchronization) among the agents but can only be used in synchronous settings. (see [10, 11, 15]).

Knowledge of the topology Different assumptions on the agents' knowledge of the topology have an impact on solutions of some of the problems mentioned above, for example, the BHS problem. In [18], Dobrev et al. present three types of topological knowledge in an asynchronous arbitrary network and show the results of the BHS problem based on different setting of the topology knowledge.

Other capabilities In some studies, agents are endowed with visibility, which means that they can see whether or not their neighbouring nodes are clean or contaminated (see

[22, 23]). For example, it is observed that the visibility assumption allows to drastically decrease the time and move complexities in tori, chordal rings and hypercubes when dealing with the IC problem. More specifically, in chordal ring $C_n\{d_1 = 1, d_2, ..., d_k\}$, the number of agents, time and moves required in the local model (i.e., when the agents do not have visibility outside the node they reside in) are $(2d_k+1)$, $3n-4d_k-1$, $4n-6d_k-1$ respectively, while in the visibility model, they decrease to $2d_k$, $\left\lceil \frac{n-2d_k}{2(d_k-d_{k-1})} \right\rceil$, $n-2d_k$. In tori, the number of agents, time and moves required are, respectively, $2h+1$, $hk-2h$, $2hk-4h-1$, while in the visibility model they are $2h$, $\lceil \frac{k-2}{2} \rceil$, $hk-2h$, respectively. The authors also compare the complexity of both models in hypercubes and propose an algorithm that requires Θ $(\frac{n}{\sqrt{logn}})$ agents and $O(n \log n)$ moves, and another algorithm in the visibility model that requires $\frac{n}{2}$ agents and $O(n \log n)$ moves.

In [5], the concept of *k-hop visibility* is presented. The agents have *full topology* knowledge if each of them have a map in their memory of the entire network including the identities of the node and the labels of the edges. An agent has *k-hop visibility*, when at a node v the agent can see the k-neighbourhood $N^{(v)}$ of v, including the node identities and the edge labels. Note that *Diam-hop* visibility is equivalent to full topological knowledge.

Another interesting capability of agents is *cloning*, which is introduced in [23]. Cloning is the capacity for an agent to create copies of itself. In [23] it is also discussed how the combination of different capacities allows different optimal strategies for the IC problem in the hypercube. For example, a time and move optimal algorithm is devised when visibility and cloning are assumed or when cloning and synchronicity are assumed and the model is local. However, the time and move-optimal strategy is obtained at the expense of increasing the number of agents. The last capability of agents that has been discussed is *immunity*: a node is immune from recontamination after an agent departs under some conditions. Two kinds of immunity have been proposed: local and temporal. In local immunity, (see [21, 37], the immunity of a node depends on the state of its neighbouring nodes. More specifically, a node remains clean after the departure of an agent until more than half of its neighbours are contaminated. In the temporal immunity, a node is immune for a specific amount of time. The node remains clean until this time expires and it becomes recontaminated if at least one of its neighbours is contaminated. In models without the immunity assumption, a node becomes recontaminated if it has at least one contaminated neighbours.

2.4 Black Virus Decontamination

2.4.1 Overview

The BVD problem is first introduced by Cai at al. in [5]: A black virus is a extraneous harmful process endowed with capabilities for destruction and spreading. The location of the initial BV(s) is known a priori. Like a BH, a BV destroys any agent arriving at the network where it resides. When that happen, the clones of the original BV spread to all the neighbouring nodes and remain inactive until an agent arrives. The BVD problem is to permanently remove any BVs in the network using a team of mobile agents. In [5] the strategy to decontaminate a BV is to surround all its neighbouring nodes and send an agent to each of them. In this case, the node where the original BV resides is clean and all its clones are destroyed by the guarding agents in its neighbouring nodes. The authors have presented different protocols in various topologies: q-grid, q-torus, hypercudes ([5]), and arbitrary graph [6]. A basic idea of implementing the decontamination has also been proposed by assuming that the timing is asynchronous, in which case the whole decontamination process is divided into two part: "shadowed exploration"and "surround and eliminate". In order to minimize the spread of the virus, the authors use a "safe-exploration" technique which is executed by at least two agents: the "Explorer Agent" and the "Leader Explorer Agent". Both agents initially resides at a safe node u, called the *homebase*. The Explorer Agent moves to a node v to explore it and it needs to return to node u to report the node v is safe. The "Leader Explorer Agent" determines if the node v

is safe or not by the arrival of the "Explorer Agent". If node v is safe, both agents move to node v. For the purpose of insuring monotonicity, at any point in time the already explored nodes must be protected so that they are not be recontaminated again. After the BV is detected, the "surround and eliminate" begins. In this phase, some agents are employed to surround the new-formed BVs (the clones of the original BV) then some agents are sent to the clones to permanently destroy them. This is called the "Four-step Cautious Walk" and is widely used in BVD problem with synchronous setting. Also, BVD problem in chordal ring has been discussed in [1].

2.4.2 BVD in different topologies

Protocols to solve the BVD problems in grid are BVD-2G and BVD-qG which deal with BVD problems in 2-dimensional grid (meshes) and q-dimensional grid respectively. BVD-2G performs a BV decontamination of a 2-dimensional grid of size n using $k = 7$ agents and 3 casualties (i.e., losses of agents), within at most $9n + O(1)$ moves and at most $3n$ time. While protocol BVD-qG performs a decontamination of a q-dimensional grid of size $d_1 \times d_2 ... \times d_q$ using $3q + 1$agents and at most $q + 1$ casualties, within at most $O(qn)$ moves and at most $\Theta(n)$ time. The Algorithm to decontaminate the BV in a q-dimensional torus, called BVD-qT uses $4q$ agents with 2 casualties with at most $O(qn)$ moves and $\Theta(n)$. Protocol BVD-qH is to perform a BV decontamination of a q-hypercube using $2q$ agents and q casualties with at most $\mathrm{O}(n \log n)$ moves and $\Theta(n)$. In arbitrary graphs (see [6]), two protocols are presented: Greedy Exploration and Threshold Exploration. In these two protocols, 2Δ agents are needed and both of the protocols are worst-case optimal with respect to the team size, where Δ represents the maximum degree in G. Also, both of the strategies cause casualties of Δ and need the number of movements of $O(\Delta^2 n)$. Though the protocols are described for a synchronous setting, they easily adapt to asynchronous ones with an additional $O(n)$ moves for the coordinating activities. An advantage of these protocols is that the agents can use only local information to execute the protocol. Another interesting fact based on these two protocols is that both GREEDY and THRESHOLD produce an optimal acyclic orientation of the graph rooted in the homebase.

I like tables. I would add summary tables for the results of Cai, as well as for the results of Madawi

In [1], solutions for BVD in chordal ring are discussed. The solutions are designed for different kinds of chordal rings: double loops, triple loops, consecutive-chords rings and finally general chordal ring. In double loops, three strategies are proposed with an upper bound of moves of $4n-7$ and a maximum of 12 agents employed. In triple loops, two classes of chordal rings are discussed: $C_n(1,p,k)$ and $C_n(1,k-1,k)$. In any triple loop $C_n(1,p,k)$, a maximum of $5n-6k+22$ moves and 24 agents are needed for the decontamination while in any triple loop $C_n(1,k-1,k)$, a maximum of $5n-7k+22$ moves and 19 agents are needed. Finally, in the consecutive-chords ring, a maximum of $(k+2)n-2k-3$ moves and $4k+1$ agents are needed. Decontamination strategies are described in synchronous settings, but with a cost of $O(n)$ moves the strategies can be used in asynchronous settings as well.

Chapter 3

Definitions and Terminology

In this Chapter, we introduce the model, the problem, and some terminology that will be used in the rest of the thesis. We also describe a general high level technique that is common to most of our solutions.

3.1 Model

3.1.1 Network, Agent, Black Virus

Network. The environment in which mobile agents operate is a network modelled as a simple undirected connected graph with $n = |V|$ nodes (or sites) and $m = |E|$ edges (or links). We denote by $E(v) \subseteq E$ the set of edges incident on $v \in V$, by $d(v) = |E(v)|$ its degree, and by $\Delta(G)$ (or simply Δ) the maximum degree of G. Each node v in the graph has a distinct $id(v)$. The links incident to a node are labelled with distinct port numbers. The labelling mechanism could be totally arbitrary among different nodes; without loss of generality, we assume the link labelling for node v is represented by set $l_v = 1, 2, 3, ..., d(v)$.

Agents. A group of mobile agents is employed to decontaminate the network. An agent is modelled as a computational entity moving from a node to neighbouring node. More than one agents can be at the same node at the same time. Communication among agents occurs when they are at the same node; there are no a priori restrictions on the amount of exchanged information. When the agent arrives at a node, it can leave a message on that node and read a message on that node. Information on the nodes can be set at the beginning of the exploration (writing the information on a whiteboard), so when an agent reaches a node, it can update the information on the whiteboard or update its own memory by learning the information on the whiteboard. Also, we assume that all the agent' s moves follow the same clock. Also, unless otherwise stated, agents are endowed with 1-hop visibility, which means that, at a node v, an agent can see the labels of the edges incident to it and the identities of all its 1-hop neighbours. Also, it can see whether or not there are agents there but cannot communicate with each other.

Black Virus. In G there is a node infected by a black virus (BV) which is a harmful process endowed with reactive capabilities for destruction and spreading. The location of the BV is not known at the beginning. It is not only harmful to the node where it resides but also to any agent arriving at that node. In fact, a BV destroys any agent arriving at the network site where it resides, just like a black hole. Instead of remaining static as the black hole, the BV will spread to all the neighbouring sites leaving the current node clean. The clones can have the same harmful capabilities of the original BVs (fertile) or unable to produce further clones (sterile). A BV will be destroyed if and only if the BV arrives at a node where there is already an agent. Thus, the only way to eliminate the BV is to surround it completely and let an agent attack it. In such a situation, the attacking agent will be destroyed while the clones of the original BV will be permanently eliminated by the agents residing the neighbouring nodes of the original node. We assume that at the same node, multiple BVs (clone or original) are merged. More precisely, at any time, there is at most one BV at each node. Another important assumption is that when a BV and an agent arrive at an empty node at the same time, the BV dies and the agent survive remaining unharmed.

Also, in this thesis, we assume that when a BV is triggered, its clones instantaneously spread to all its neighbour. That is, when a BV is triggered by an agent at $\mathrm{T}(t_i)$, then at the same time $\mathrm{T}(t_i)$, all the neighbours of this agents (if any) receive its clones. The reason we make this assumption is that when we try to eliminate the new formed BVs parallelly in the chordal ring and assume that it takes one unit of time for both the agents and the BV to move from one node to another, then we are faced with a tricky situation: if the sites of two (or more) agents are connected, after these two clones are triggered(we send an agent to each of them to permanently destroy them), one of their clones spread to another site and since the agents sent to contaminate them die, these two sites are empty when the second round clones arrive, which make the decontamination invalid. While with the assumption, the tricky situation can be easily solved: after the two clones are triggered, one of their clones spread to another site and both the original clone and the second round clone are destroyed by the agent.

Summarizing, there are five possible situations when an agent arrive at a node v:

- agents arrive at a node which is empty or contains other agents, they can communicate with each other and the node v is clean.
- agents arrive at a node which contains a BV, the clones of the BV (BVC) spread to all the neighbours of v and the agent dies, leaving node v clean.
- A BVC arrives at a node which is empty or there is already a BV: the node becomes/stays contaminated; it merges with other BVs.
- BVCs arrive at a node v which contains one or more agents, the BVCs are destroyed but the agents are unharmed.
- A BVC and an agent arrive at an empty node at the same time, the BVC dies while the agent remains unharmed.

3.1.2 Problem and Cost Measures

The Black Visur Decontamination (BVD) problem is to permanently remove the BV and its clones from the network using a team of mobile agents starting from a given node, called homebase (HB).

In this thesis, the cost measures are the following:

- the *spread* of the BV: number of nodes that receive a BV or a BVC. This measure also evaluates the loss of agents (or *casualties*);
- the *size* of the team, i.e, the number of agents employed by the solution;
- the *total working time* (TWT), calculated by multiplying the *size* of the team and the *time* cost of the solution. We propose TWT to compare more fairly the time of two protocols when the number of agents is different.

3.1.3 Monotonicity and Synchrony

A desirable property of a decontamination protocol is to prevent the nodes that have been explored or cleaned by mobile agent from being recontaminated, which would occur if the clones of the BV are able to move to an empty already explored node. A BVD protocol with such property is called *monotone*. The monotonicity property is a necessary condition for spread optimality.

Asynchrony refers to the execution timing of agent movement and computations. The timing can be *Synchronous* or *Asynchronous*. When the timing is synchronous, there is a global clock indicating discrete time units; it takes one unit for each movement (by agent or BV); computing and processing is negligible. When we have asynchronous agents, there is no global clock, and the duration of any activity (e.g., processing, communication, moving) by the agents, the BV, and its clones, is finite but unpredictable. In this thesis, all our protocols work in synchronous setting.

3.2 General strategy

Following the solution of sequential case, we decomposed the BVD process into two separate phase: *Shadowed Exploration* and *Surrounding and Elimination*. The task of the first phase is to locate the BV and the second phase is to decontaminate the BV and its clones. Additionally to these two basic phases, we have and *Initialization* phase, which is to properly deploy the agents before starting the execution of the protocol. In fact, since the network is explored in parallel, the initial arrangement of the agents is crucial to successfully execute the protocol.

Phase1: Shadowed Exploration. The agents employed are divided into two groups: shadowing group and exploring group, and the number of agents in the two group is the same. For convenience, we call the agents in the shadowing group the shadow agents (SA) and those in the exploring group the exploring agents (EA) (one EA is always accompanied by one SA). As the name indicates, agents in the exploring group explore the network and agents in shadowing group follow the exploring agents protecting the nodes that have been already explored.

Phase2: Surrounding and Elimination. In this phase, since we already know the position of the BV, we employ agents to surround all the neighbours of the BVCs. Once all the agents arrive at the proper positions (all the neighbours of the BVCs are guarded), we employ another group of agents (the number of them is equal to the number of BVCs) to move to the BVCs, thus permanently some from the exploring group because in this way we can save the total number of agents used in the whole process. Note that not all the agents are informed when the BV is detected. More specifically, only the agents that receive the clones know the existence of the BV, while the other agents keep moving in the network. In some simple topologies, such as meshes and tori, the second phase begins when the BV is detected since the number of agents are sufficient to proceed the second phase. In some more complicated topologies, for example, in the chordal ring which we discuss later, we take some other measures to call back enough agents to finish the second phase.

3.3 Summary

In this chapter, the basic terminology and the model for the Black Virus Decontamination problem are introduced. Monotone property is the necessary condition for the spread optimality and should be obeyed.The general strategy for the Black Virus Decontamination is also presented and the basement of our parallel strategies.

Chapter 4

Parallel Black Virus Decontamination in Meshes and Tori

4.1 Introduction

In this Chapter, we discuss parallel strategies for the BVD problem in grids and tori. In the sequential strategy (*BVD-2G*) for 2-dimensional grids, an "explorer agent" and a "leader explorer agent" are sent to explore the graph and locate the BV. They traverse the mesh in a snake-like fashion, column by column, following "cautious walk". In the sequential protocol (*BVD-qG*) for q-dimensional grids, the grid is partitioned into $d_1 \times \ldots \times d_{q-2}$ 2-dimensional grids of size $d_{q-1} \times d_q$, and each 2-dimensional grid is explored using the shadowed traversal technique as described in the 2-dimensional grids. Similarly, in the protocol $(BVD-qT)$ for q-dimensional torus, the torus is partitioned into $d_1 \times \ldots \times d_{q-1}$ ring of size d_q. The exploration procedure traverses a ring and, when back to the starting point, proceeds to another ring, with a neighbouring starting point. After locating the BV, agents surround the new formed BVs sequentially and eliminate them. These strategies are simple to follow, but at the same time they are not time-efficient. We now consider the situation when more than two agents are allowed to participate in the exploring phase and we focus on decreasing the time cost in the exploring phase and the number of casualties; that is, we focus on how to design a new strategy so that the we are able to reach the destination faster, but with an acceptable cost in terms of number of agents.

at his point it is not clear what you mean by “array”, do you mean a compact area of consecutive/adjacent nodes ? I add the explanation below. The general idea is simple, we will employ a group of agents and place them in a compact area of consecutive or adjacent nodes (for convenience, let us call it “array”) at the beginning. Informally, in the shadowed exploration of our strategy in 2-dimensional grid(*PBVD-2G*), q-dimensional grid (*PBVD-qG*) and tori (*PBVD-qT*), the agents that are employed to explore the graph stay in that array and this “array of agents” traverse that graph in the shadowed exploration. Note that, after the BV is triggered, not all the agents automatically enter the elimination phase but only the ones that know the existence of the BV. However, in some cases, the number of agents that know the existence of BV is not sufficient for surrounding and eliminating the BVs. For this reason, in the elimination phase, our strategies employ some agents that know the existence of the BV (because they receive the clones of the original BV) to notify some other agents to participate in the elimination phase as well. We study the number of agents, the time cost, the number of movements and casualty, comparing them with the ones of the corresponding sequential strategy.

4.2 Parallel BV Decontamination of Grids

4.2.1 Base Case: 2-Dimensional Grid

A 2-dimensional grid (which is a mesh) of size $d_1 \times d_2$ has $n = d_1 \times d_2 (d_1 > 2, d_2 > 2)$ nodes. Without loss of generality, let $d_1 < d_2$ and let the nodes of M be denoted by their column and row coordinate (x_1,x_2), $1 \leq x_1 \leq d_1$, $1 \leq x_2 \leq d_2$. Observe that in a mesh, we have three types of nodes: *corner* (entities with only two neighbours), *border*(entities with three neighbours), and *interior*(with four neighbours). Our strategy follows two phases: shadowed exploration and elimination. In the first phase, the network is traversed until the location of the BV is determined. That location is found after the visit, at which time all the unprotected neighbours have become BVs. Note that in $PBVD-2G$, there is only one new formed BV. In the second phase, the new formed BV is surrounded and permanently eliminated. Note that when we say that the second phase starts, we actually mean that those agents knowing the existence of BV start to surround the BV, or notify some other agents, and then eliminate the BV, but not that all the agents enter this phase. There are two significant differences between $PBVD-2G$ and the sequential strategy: mainly the number of agents employed in the exploration phase and the route of agents in the exploration phase. We also describe the routes of agents in the elimination phase.

Shadowed Exploration Phase

As we mentioned above, we should place the agents in a specific array at the beginning and then let them explore the graph. Let us now consider how to arrange them at the beginning and how to design the routes for them to explore the graph. We prefer to place the agents at the borders (or the corners) of the mesh because in this way we can reduce the casualties. For the same purpose of reducing the casualties, we prefer to arrange all the agents in an array so that when one of the exploring agent triggers the BV, the exploring agents and shadowed agents guide as many neighbours of the BV as possible. In other words, we want all the agents to explore the graph in one direction rather than from different directions. With these two principles, our strategy in the shadowed exploration is the following: given a specific number of agents, we place them in one border of the mesh and if there are more agents, we place the extra agents in the next row, and so on. Then we design routes for the agents so that at any time they move to the same direction to explore the graph.

Monotonicity is a principle that we should obey in the whole process, which means one exploring agent should be followed by at least one shadowed agent, so the number of agents in the exploration phase should be at least twice the number of the exploring agents. To guarantee the monotonicity and the two principles, we should employ $2a\,(a \in \mathbb{Z}^+)$ agents in this phase and place a of them in one of the border and the others in the second line paralleling to the border. Let us now consider the number of agents we should employ. When $a = 1$, the arrangement is actually the same as in the sequential case. Since we want to explore the graph in parallel, we start with $a = 2$, in which case, the initial arrangement would be as depicted in Figure 4.1:

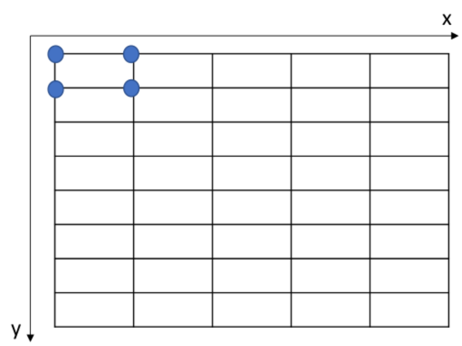


Figure 4.1: Arrangement of agents at the beginning(when $a = 2$)

Let us now consider the routes for the agents, which will result to be a snake-like path. For convenience, we assume that all the agents move at time $t_i\,(i \in \mathbb{Z}^*)$??? because some time should be reserved for the coordination after the BV is triggered.???Let $v = (x, y)$ be the node under exploration, with $1 \leq x_i \leq d_1$, $1 \leq y_i \leq d_2$. We define a boolean value called *Vertical Moving Mark* (VMM) for every agent as follows: the value VMM is initialized at zero and every time the agent moves SOUTH, it switches to (VMM+1)mod 2. For example, if an agent continues to move SOUTH the VMM will keep alternating between 0 and 1, essentially indicating the parity of the row.

Every agent holds two VMMs in its memory (say VMM_1 and VMM_2). The original value of VMM_1 is 0; the original value of VMM_2 is 0 for the agents residing at node $(1, y)$

and 1 for the ones residing in node $(2, y)$ $(1 \leq y \leq a)$. We now define the action of the $2a$ agents:

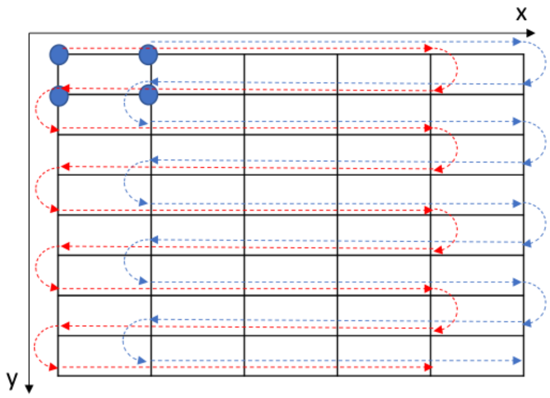
- Let $b = a - 1$. these should be local rules, how can the individual agent know that all values are zero. You have to express the rule in a different way, not replying on this type of info. (make some modification, please check)

  When an agent' s VMM_1 is "0": then it moves EAST when $x \neq d_1 - 1$ and move SOUTH for b steps when $x = d_1 - 1$ when its VMM_2 is equal to "0" ; or it moves EAST when $x \neq d_1$ and move SOUTH for b steps when $x = d_1$ when its VMM_2 is equal to "1".

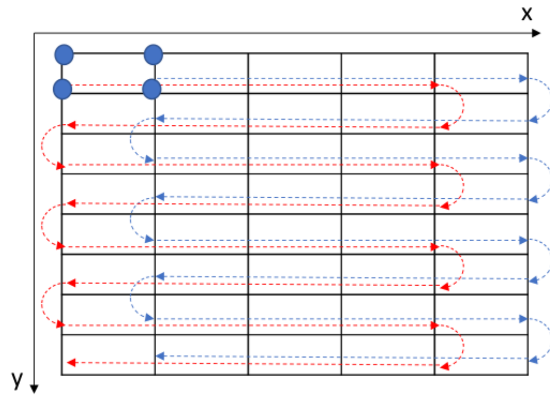
  When an agents' VMM_1 is "1": then it moves WEST when $x \neq 2$ and move SOUTH for b steps when $x = 2$ when its VMM_2 is equal to "0" ; or it moves WEST when $x \neq 1$ and move SOUTH for b steps when $x = 1$ when its VMM_2 is equal to "1" .

- An agent moves only to a node that it has not explored yet. (Note that when residing at a node, an agent is able to know whether it has explored the neighbours of that node or not).

Informally, the resulting routes of agents are snakelike routes. When $a = 2$, the routes of agents are shown as Figure 4.2. In order to show the routes more clearly, we present the routes of agents in different line respectively.



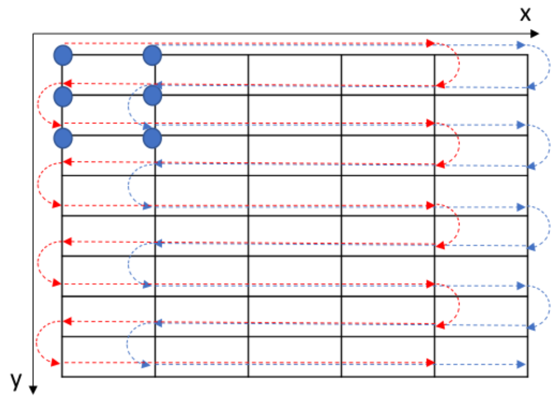
(a) Routes of agents in the first line when $a = 2$



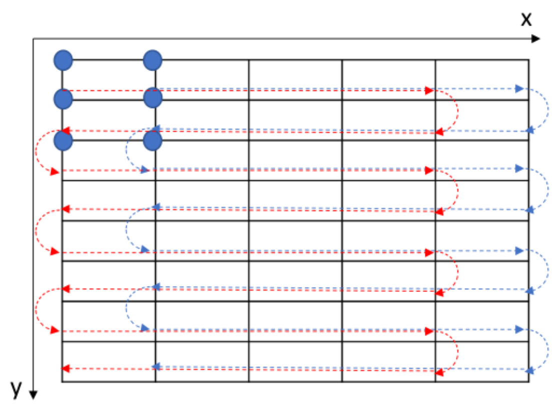
(b) Routes of agents in the second line when $a = 2$

Figure 4.2: Routes of agents when $a = 2$

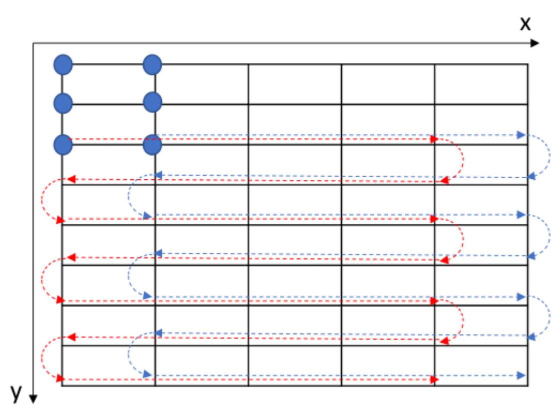
Figure 4.3 shows the routes of agents when $a = 3$.



(a) Routes of agents in the first line when $a = 3$



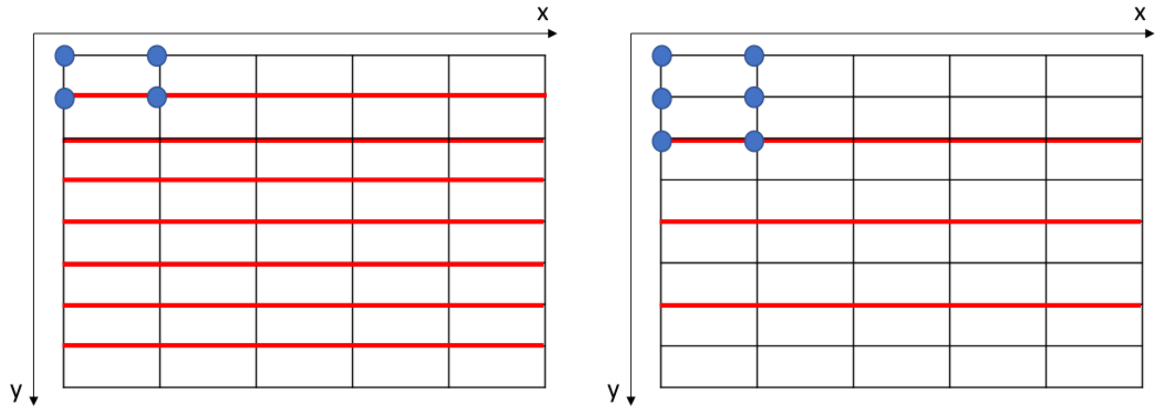
(b) Routes of agents in the second line when $a = 3$



(c) Routes of agents in the second line when $a = 3$

Figure 4.3: Routes of agents when $a = 3$

We can easily observe that some rows of the grid are traversed twice for the purpose of avoiding the explored nodes being contaminated (as shown in Fig 4.4).



(a) Rows that are double traversed when $a = 2$ (b) Rows that are double traversed when $a = 3$

Figure 4.4: Rows that are double traversed (marked with red rows)

We can see that when the grid is fixed (so d_1 is fixed), the number of double traversed rows decrease as a increases, which is easy to image. Let us denote by r the number of rows that are double traversed, then $r = \lceil \frac{d_1 - a}{a-1} \rceil$ where d_1 is the number of rows of the grid. Informally, r indicates the time that is spent in the exploration phase, and we can reduce this time by employing more agents. As we can see from the equation, when $a = d_1$, then $r = 0$, which means if we employ $2d_1$ agents to explore the graph, none of the rows are traversed twice.

We now discuss the strategy when we employ $2d_1$ agents, and this strategy can be easily modified to fit the situation where we employ less than $2d_1$ agents.

Initially, $2d_1$ agents are placed at the first two columns at t_0 and we place another agent at the top and bottom of the first column (we shall describe their roles in the following part). More specifically, their coordinates are (1, x_i) and (2,x_i) where $1 \le x_i \le d_1$. The agents residing in the first column are in the shadowing group while the agents residing in the second column are in the exploring group. If the BV resides in a node of the first column, then all of its clones are destroyed. If the BV resides in a node in the second column, then the elimination phase begins. It is obvious that if the BV does not reside in any node in the first column, then an agent in the exploring group should be destroyed when the BV is exposed. Let us assume that the we starts at t_0. Agents residing in nodes of the second column move EAST at the beginning of t_i, $i = 0, 1, \ldots, d_2 - 1$. More precisely, the agent located in (x, y) moves to $(x + 1, y)$ at the beginning of t_i, $i = 0, 1, \ldots, d_2 - 1$. Agents residing in the first column simply follow the node in the second column (see Fig.4.5). When one of the nodes in the second column is destroyed by a BV, the second phase starts.

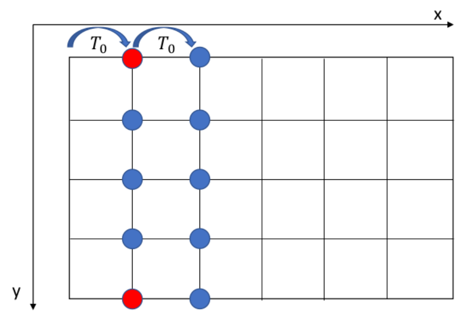


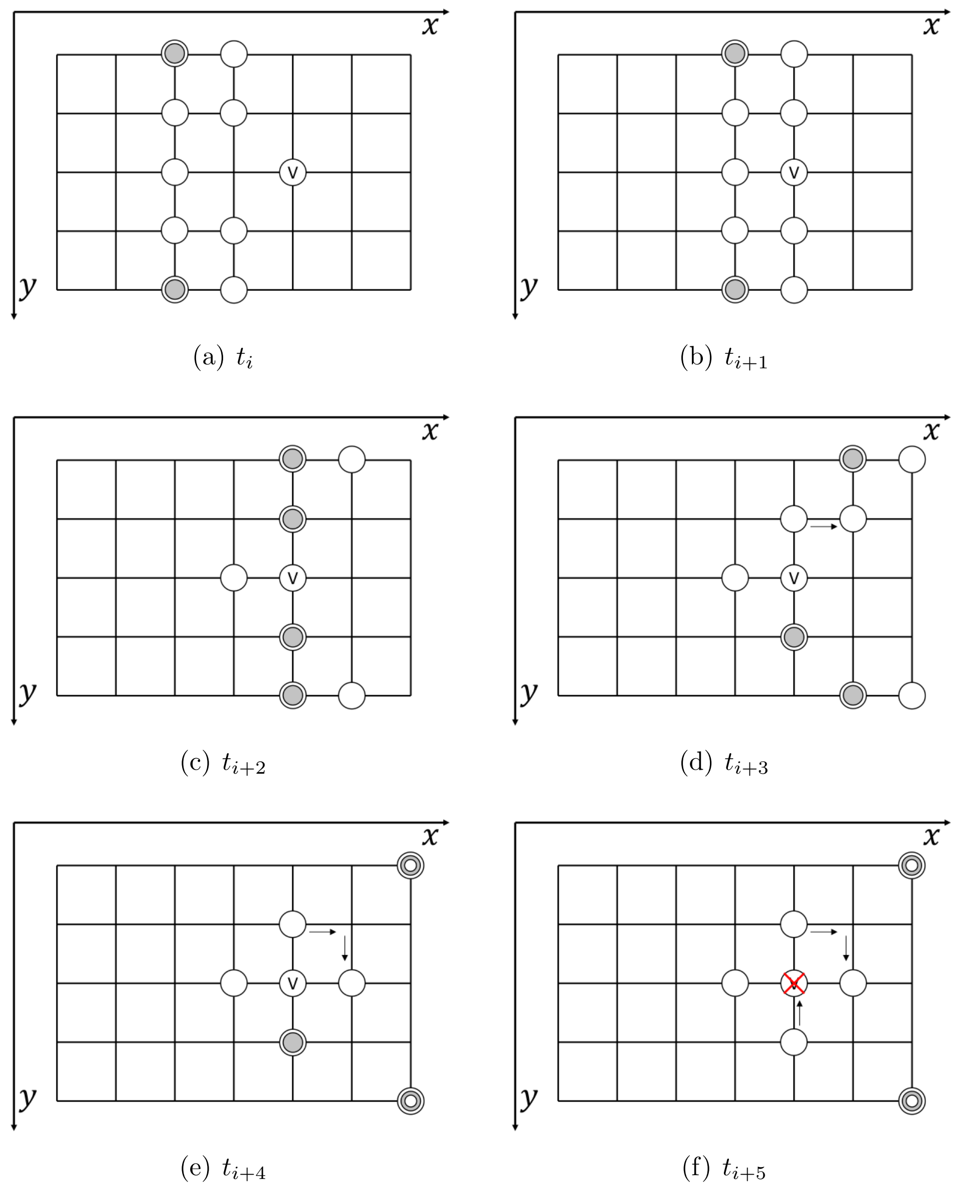
Figure 4.5: Agents move at time t_0. (The red node indicates that there are two agents residing there

Elimination Phase

The elimination begins when one of the nodes in the second column is destroyed by a BV (let us say, at time t_i). No matter where the BV is, there are always three agents residing on its north (if the BV is not in the first row), west and south (if the BV is not in the last row), so only one BV clone survives. In another words, only one node becomes BV after the BV has been triggered. Observe that in the parallel strategy, not all agents participate in the elimination phase automatically when the BV is explored because only the ones that receive the clones of the original BV and those that are notified by other agents can participate in the elimination phase. So, in some situations, agents that receive the BV clone should notify other agents to participate in the elimination (cases 1,2 and 3 below). In one particular situation (case 4 below), we instead use the agent that we take along the way (called following agent) to complete the elimination phase. Let the node where the surviving clone reside be (x, y). We have four different situations depending on the location of the new formed BV, each situation corresponding to a different route taken in the elimination phase.

- Case 1: When $2 < x < d_1$, $1 < y < d_2 - 1$ (an interior node becomes a new formed BV), then the agents residing in node $(x-1, y+1)$, $(x-1, y-1)$ and $(x-2, y)$ (say a, b, c) receive a BV clone at time t_i, and they know the location of the original BV and also the new formed BVs. After they receive the BV clone, these agents move EAST at t_{i+1} for one step (for example, to node $(x, y+1)$, $(x, y-1)$ and $(x-1, y)$) and stop. Note that other agents including the ones residing in node $(x-2, y+1)$ and $(x-2, y-1)$ (say agent d and e) at t_i do not know the existence of the BV so they keep moving EAST and arrive at nodes $(x, y+1)$, $(x, y-1)$ at the end of t_{i+2} when they meet agent a and b respectively. Agent a and b inform them of the location of the new formed BV and the routes of agents d and e are as follow:
  route of d: $(x, y+1)(at\ t_{i+2}) \rightarrow (x+1, y+1)(at\ t_{i+3}) \rightarrow (x+1, y)(at\ t_{i+4})$.
  route of e: $(x, y-1)(at\ t_{i+2}) \rightarrow (x, y)(at\ t_{i+5})$.

  The routes of agents are showed in Fig.4.6 where “one circle” indicates that there is one agent residing here; “two circle” indicates that there are two agents residing here; “three circle” indicates that there are three agents residing here.

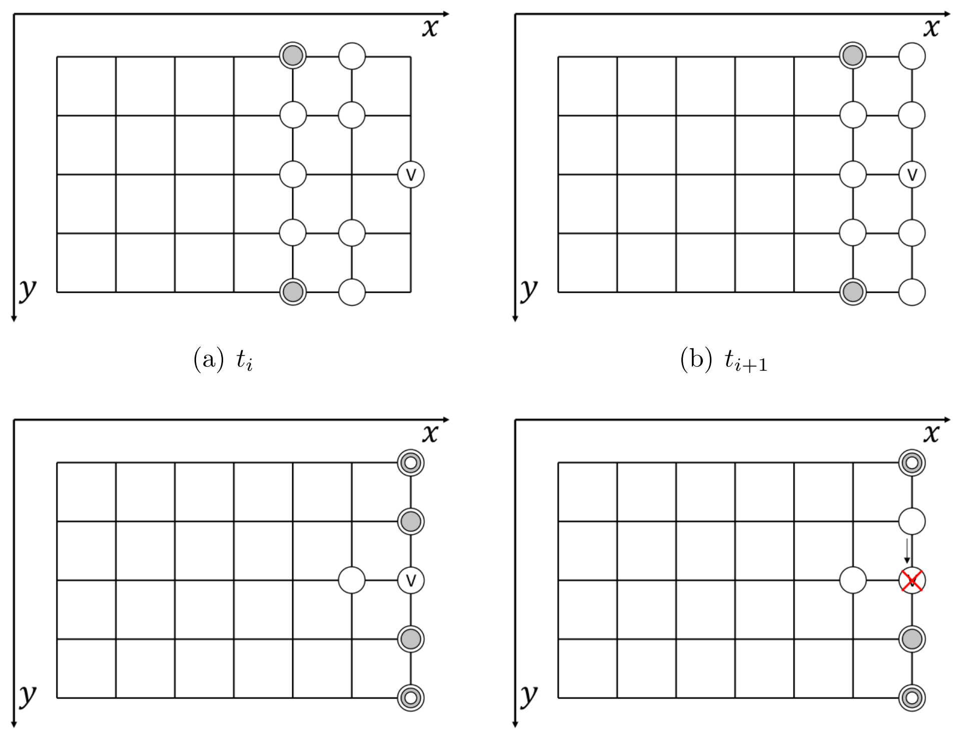


(a) t_i (b) t_{i+1} (c) t_{i+2} (d) t_{i+3} (e) t_{i+4} (f) t_{i+5}

Figure 4.6: Arrangement of the agents in elimination phase when the new formed BV resides in a interior node

- Case 2: When $x = d_1$, $2 < y < d_2 - 1$ (a border node becomes a new formed BV), then the agents residing in node $(x-1, y+1)$, $(x-1, y-1)$ and $(x-2, y)$ (say a, b, c) receive a BV clone at time t_i. As above, they move EAST for one step and stop. The agents residing in nodes $(x-2, y+1)$ and $(x-2, y-1)$ (say a, b) at t_i have no knowledge of the BV, so they keep moving and arrive at nodes $(x, y+1)$

and $(x, y-1)$ at t_{i+2} when they are informed of the location of the new formed BV. One of the agents should move to the new formed BV to decontaminate it while the other stops moving at t_{i+3}. In order to avoid conflict, we always employ the agent who observes that the BV is in its SOUTH (say agent a) to move to the new formed BV (see Fig 4.7).



(a) t_i (b) t_{i+1} (c) t_{i+2} (d) t_{i+3}

Figure 4.7: Arrangement of the agents in the elimination phase when the new formed BV resides in a border node (when $x = d_1$)

- Case 3: When $2 < x < d_1$, $y = 1$ or $y = d_2 - 1$ (a border node becomes a new formed BV). For convenience, we only discuss the situation when $y = 1$(the solution can be easily modified to fit the scenario when $y = d_2 - 1$). In this case, agents residing in nodes $(x-1, y+1)$ and $(x-2, y)$ (say a, b, c, where c is the following agent) receive a BV clone at time t_i. Agents a, b and c move EAST for one step and arrive at node $(x, y+1)$ and node $(x-1, y)$ at t_{i+1}. The agent in node $(x-2, y+1)$ (say, d) does not know the existence of the BV, so it keeps moving arriving at node $(x, y+1)$ at

$t_i + 2$. After that, the routes of the agents c (the following agent) and d are described below:

route of d: $(x, y+1)(at\ t_{i+2} \rightarrow (x+1, y+1)(at\ t_{i+3} \rightarrow (x+1, y)(at\ t_{i+4})$.

route of c: $(x-1, y)(at\ t_{i+2} \rightarrow (x, y)(at\ t_{i+5})$.

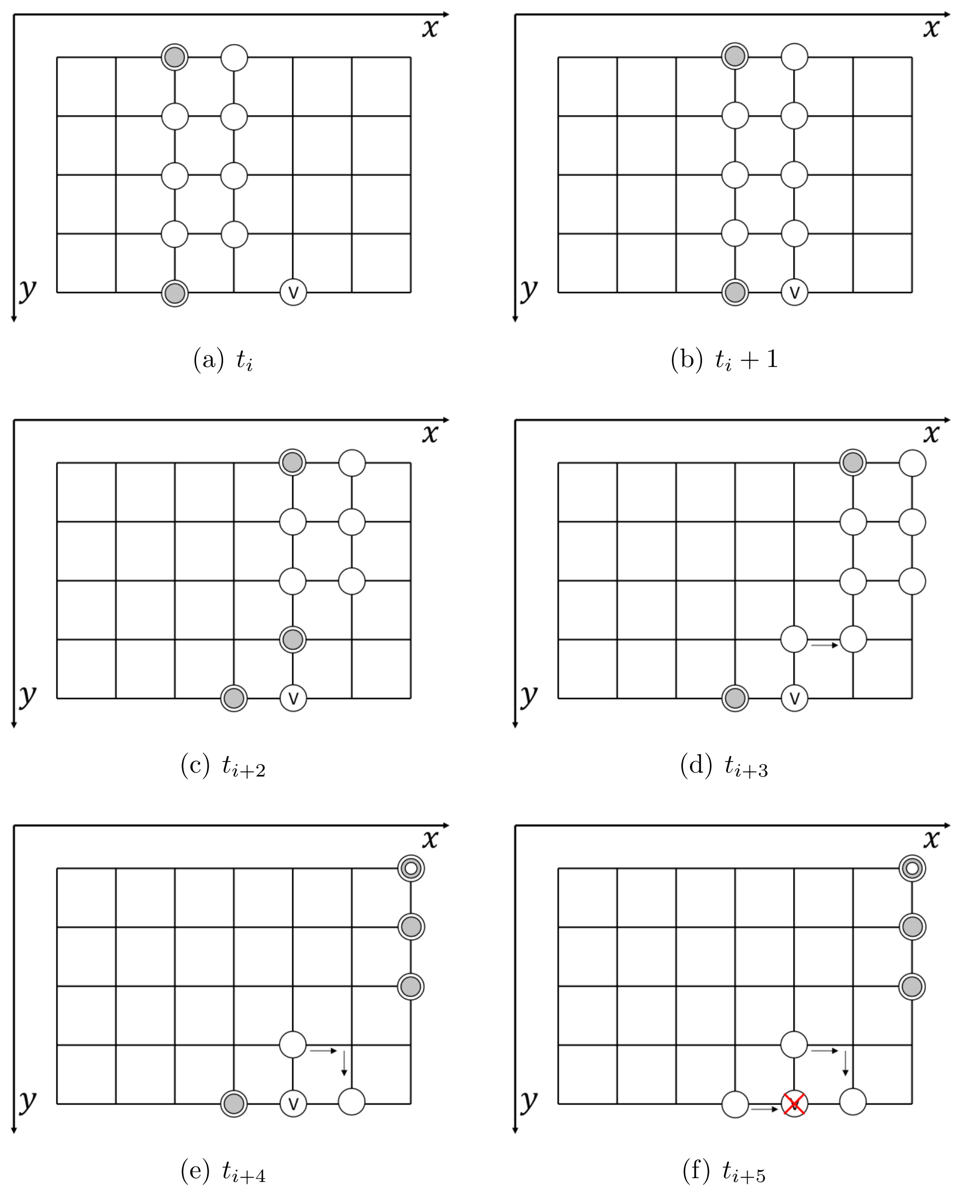
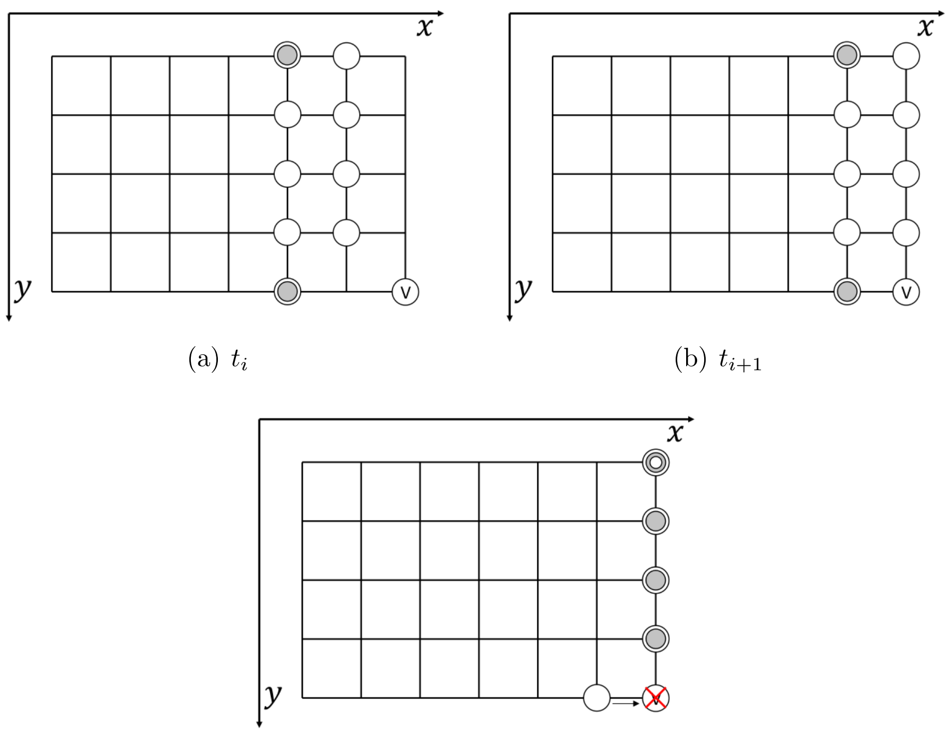


Figure 4.8: Arrangement of the agents in the elimination phase when the new formed BV resides in a border node(when $y = 1$ or $y = d_2 - 1$)

- Case 4: When $x = d_1$ and $y = 1$ or $y = d_2 - 1$ (a corner node becomes a new formed

BV). For convenience, we only discuss the situation when $y = 1$ and with some simple modification, the strategy can fit the scenario when $y = d_2 - 1$. In this case, agents residing in node $(x-1, y+1)$ and $(x-2, y)$ (say, a, b) receive a BV clone at t_i. Both of them keep moving for one step arriving at nodes $(x, y+1)$ and $(x-1, y)$ at t_{i+1}. Then agent b moves to the BV to destroy it at t_{i+2}



(a) t_i (b) t_{i+1}

(c) t_{i+2}

Figure 4.9: Arrangement of the agents in the elimination phase when the new formed BV resides in a corner node(when $x = d_1$ and $y = 1$ or $y = d_2 - 1$)

Analysis and Comparisons

Let $\mathcal{M}$ be a mesh of size $n = d_1 \times d_2$ with $d_1 = min\{d_1, d_2)\}$.

Theorem 1. *Algorithm $PBVD{-}2G$ performs a decontamination of $\mathcal{M}$ using $k = 2(d_1+1)$ agents (that is, $(2(\sqrt{n}+1)$ agents in the worst case), and incurring in at most 1 casualty.*

Proof. Let $v = (x, y)$ be the node containing the BV. When one of the agents in the exploring group moves to v, it will be destroyed and the BV will move to all neighbours of v. If $x = 1$, then the neighbours $(x, y+1)$, $(x, y-1)$ and $(x+1, y)$ are protected by agents and neighbour $(x-1, y)$ actually does not exist; if $x > 1$, then neighbours $(x, y+1)$, $(x, y-1)$ and $(x-1, y)$ are protected by agents. So when the clones of BV moves to the neighbours of v, the nodes that contain an agent will not be infected by the BV clone; this means that the BV can safely move only to the unexplored neighbours of v, which is at most one. In other words, after v is explored, at most one BV node is formed. According to our elimination strategy, the new formed BV node can be surrounded and destroyed using at most five agents: one to enter a BV and four to protect the neighbours. Since we have one new formed BV, the number of agents participating in the elimination phase is at most five. In addition to the agent destroyed by the original BV, the number of agent needed to complete the elimination phase is at least six. Since we employ $k = 2d_1 + 2$ $(d_1 \geq 3)$ which means at the beginning we have at least eight agents, so $2d_1 + 2$ agents are enough for the decontamination algorithm. In the worst case (which is the case of a square mesh where $d_1 = d_2$) the number of agent is equal to $2\sqrt{n} + 2$. □

Let us now consider the number of movements.

Theorem 2. *Algorithm $PBVD-2G$ performs a BV decontamination of $\mathcal{M}$ with at most $2n - \sqrt{n} + O(1)$ movements and at most $\sqrt{n} + 11$ in time.*

Proof. Let $v = (x, y)$ be the BV node, and let the size of the grid be $n = d_1 \times d_2$. Let us first consider the number of movements performed during the shadowed exploration. Since all the agents simply move EAST at the beginning of $T(2n)$ $(n = 0, 1, \ldots, d_2 - 1)$, the travelling distance is x for agents in the exploring group (EA) and $x - 1$ for agents in the shadowing group (SA). We have d_1 EAs and $d_1 + 2$ SAs, then we have an overall cost of at most $2x(d_1 + 1) - (d_1 + 2)$ movements for this phase. Consider now the number of movements performed for Surrounding and Elimination. In this part, we only compute the movements of the agents that participate in the Surrounding and Elimination. More specifically, we ignore the movements of the agents that do not know the existence of the

BV in the whole process. As we discussed in the Elimination phase, when the new formed BV is located in an interior node, eight movements are needed in this phase; when the newly formed BV is located in a border node (say (a, b)), then six movements are needed when $a = d_1, 2 < b < d_2 - 1$ and eight movements are needed when $2 < a < d_1, b = 1$ *or* $b = d_2 - 1$; when the newly formed BV is located in a corner node, then four movements are needed in this phase. Hence, $O(1)$ movements are performed in this phase. In total we have that the number of movements is at most $2x(d_1+1)-(d_1+2)+O(1) \leq 2\sqrt{n}(\sqrt{n}+1)-(\sqrt{n}+2)+O(1)$, which is $2n + \sqrt{n} + O(1)$.

As for the time complexity. The time required for the exploration phase is equal to the number of movements of each EA, which is d_1; the time required for the surrounding and elimination phase is at most eleven. So in total the parallel mesh decontamination algorithm terminate in time at most $\sqrt{n} + 11$. □

Table 4.1 shows a comparison between our strategy and the sequential strategy.

Table 4.1: Comparison between PBVD-2G and BVD-2G

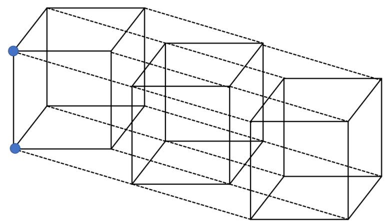
	agents	time	movements	casualties
PBVD-2G	$2(d_1 + 1)$ $(d_1 = min(d_1, d_2))$	$\sqrt{n} + 11$	$2n + \sqrt{n} + O(1)$	1
BVD-2G	7	$3n$	$9n + O(1)$	3

4.2.2 Multi-Dimensional Grid

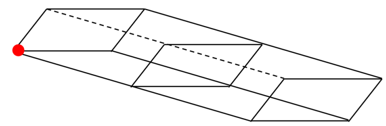
Let M be a q-dimensional grid of size $d_1 \times \ldots \times d_q$ and let each node of M be denoted by its coordinates $(x_1, \ldots, x_q)$, $1 \leq x_i \leq d_q$. The algorithm, called *PBVD-qG*, follows a general strategy similar to the one described in Section 4.2.1: a safe exploration with shadowing, followed by a surrounding and elimination phase. Our general idea is as follows: (1)

Transform the PBVD-qG problem into a BVD-qG problem (2) Use the BVD-qG strategy to solve the transformed problem.

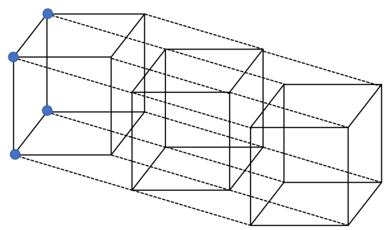
In [5], the multi-dimensional grid is partitioned into $d_1 \times \ldots d_{q-2}$ 2-dimensional grids of size $d_{q-1} \times d_q$. Each of these grids is explored using the BVD-2G: after traversing a grid in a snake-like fashion column by column, the agent returns to the starting point and from that starting point, it proceeds to another grid with a neighbouring starting point. In our strategy PBVD-qG, we use the similar exploring routes as that in BVD-G, which is the snake-like route. Additionally, we use the idea of *dimensionality reduction*. In our parallel strategy we use the term "p-dimensional agent group" to refer to a group of $d_1 \times \ldots \times d_p$ agents ($p < q$) organized in a $d_1 \times \ldots \times d_p$ grid (i.e., fully occupying a $d_1 \times \ldots \times d_p$ sub-grid of the original q-dimensional grid). Informally, a "p-dimensional agent group" can be viewed as one "large agent" and the q-dimensional grid as a "virtual" $(q - p)$-dimensional grid. Clearly, the larger p, the smaller the size of the virtual grid to be explored (see Figure 4.10 for examples).



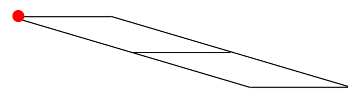
(a) The arrangement of agents on a 4-dimensional grid



(b) When we view the “one dimensional” agent as a large agent and the grid can be view as a three dimensional grid



(c) The arrangement of agents on a 4-dimensional grid

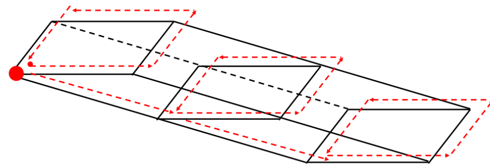


(d) When we view the “two dimensional” agent as a large agent and the grid can be view as a two dimensional grid

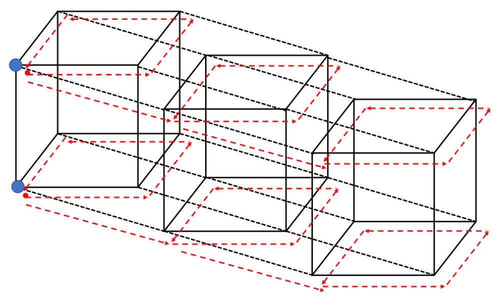
Figure 4.10: The idea of dimensionality reduction

After we transform the problem, we can use the BVD-qG to solve the PBVD-qG. Let us now have a brief review of the BVD-qG strategy: there are two agents exploring the graph: one is called exploring agent (EA) and one is called leader exploring agent (LEA). They follow the “four step cautious walk” strategy to explore the graph: before exploring a node $v = (x_1, \ldots, x_q)(1 \leq x_i \leq d_i)$ from a node u, the shadowing agents(SA) move to the already explored neighbours of v (whose coordinated can be precisely computed). When EA visits the BV node (and is destroyed there), the LEA and the SAs are aware of the location of the new BV nodes (its coordinates can be computed precisely). So, once the node v containing the BV is identified, $2q$ agents surround each node $u \in N_{un}(v)$ and an additional agent enters it destroying the BV resident there and the instances that it generates.

Assume that we employ a "p dimensional" agent group, which consists of $d_1 \times \ldots d_q$ agents with coordinates: $(x_1, \ldots, x_p, \ldots, x_q)$, $(1 \leq x_1 \leq d_1, \ldots, 1 \leq x_p \leq d_p, x_{p+1} = 0, \ldots, x_q = 0)$. Then the coordinates of the "big agent" in the $q-p$ dimensional grid are $(x_{p+1}, \ldots, x_q)$. The first p dimensional coordinates of the agents would not change in the whole exploring phase, but from that $(p+1)^{th}$ dimensional coordinates, they change as the "big agent" changes. More specifically, the coordinates of the $(p+1)^{th}$ dimension of the agents make the same change as the first dimensional coordinate of the "big agent"; the coordinates of the $(p+2)^{th}$ dimension of the agents make the same change as the second dimensional coordinate of the "big agent", and so on. Assume now that we employ four agents (two agents are viewed as the LEA in the BVD-qG; two agents are viewed as the EA in the BVD-qG) to explore the 4-dimensional grid (they traverse the grid with the same routes following "four step cautious walk"), then the routes of the agents are as below (see Fig.4.11):



(a) The route of the “big agent”(LEA in the BVD-qG)



(b) The routes of the 2 agents viewed as the “big agent”

Figure 4.11: The route of agent when exploring a $1 \times 1 \times 1 \times 2$ grid

Analysis and Comparisons We now compare the time cost, the number of movements and the casualties with different number of agents we employ (Assuming that we are in a $d_1 \times \ldots \times d_q$ q-dimensional grid). The theorems in [5] show that the protocol BVD-qG performs a BV decontamination of a q-dimensional Grid using $3q + 1$ agents with at most $q + 1$ casualties and at most $O(qn)$ movements and $\Theta(n)$ time. Based on these theorems, we now discuss the number of agents, the casualties, the time cost and the number of movement in PBVD-qG.

Let $\mathcal{M}_q$ be a q-dimensional grid of size $n = d_1 \times \ldots \times d_p \times \ldots \times d_q$.

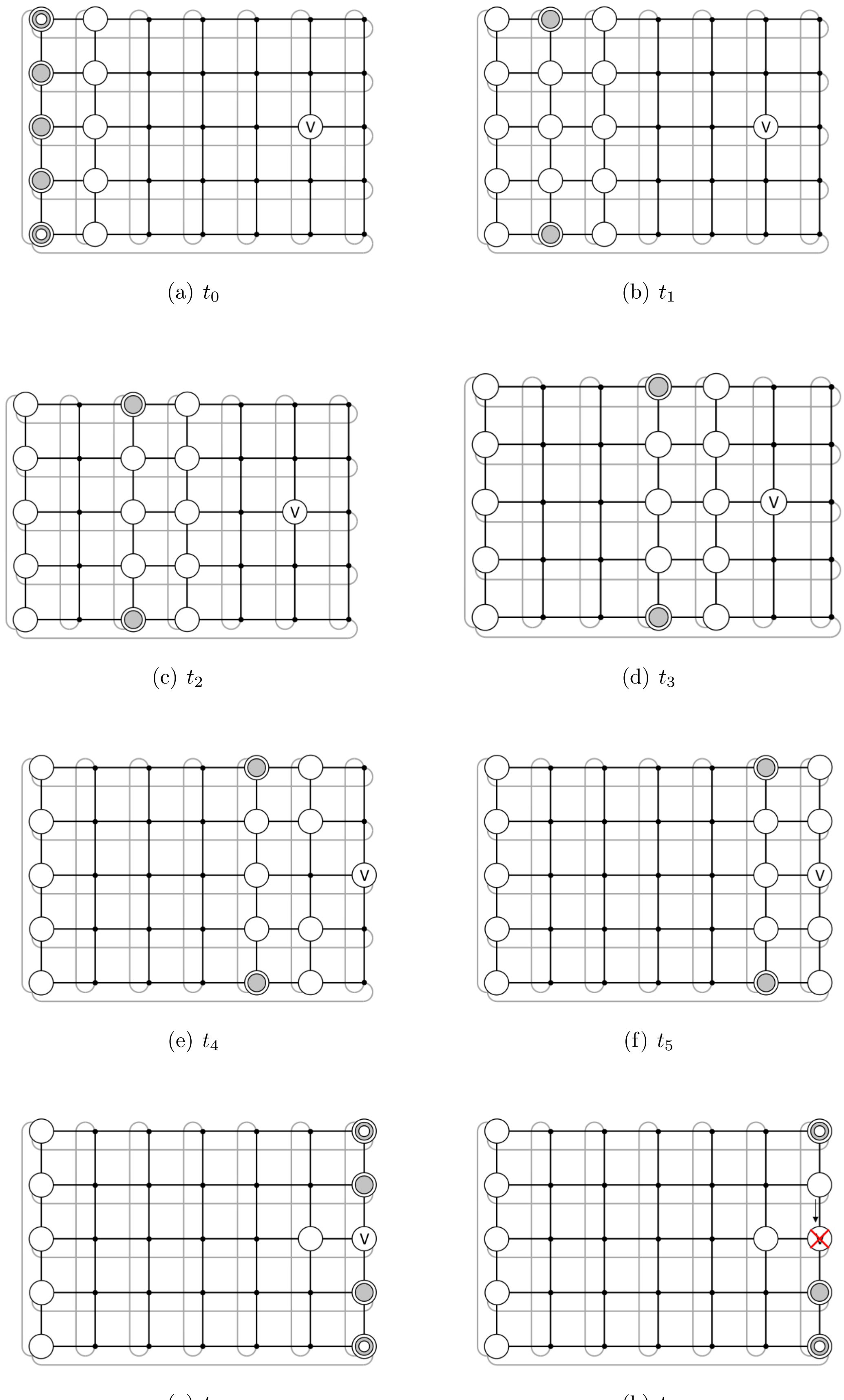
Theorem 3. *PBVD-qG performs a decontamination of $\mathcal{M}_q$ using $2 \times d_1 \times \ldots d_p (1 \leq p \leq q)$ agents and at most $q - p + 1$casualties, with at most $O(qn)$ moves and $\Theta(\frac{n}{d_1 \times \ldots \times d_p})$ time.*

Proof. In PBVD-qG, we can choose different number of agents to start the exploration, and that number results in different number of casualties. Assume that we use $2 \times d_1 \times \ldots d_p$ agents to explore the graph (with $1 \leq p \leq q$), then we can view $d_1 \times \ldots d_p$ agents as forming a "big agent". As we mentioned before, we need $d_1 \times \ldots d_p \times (3q+1)$ agents in total (every $d_1 \times \ldots d_p$ agents play the role of one agent in the BVD-qG). Now our problem changes into solving the BVD-qG problem into a $q-p$ dimensional grid with $d_{p+1} \times \ldots d_q$ agents, so the casualties and the times cost follow accordingly, by considering the same situation when we use BVD-qG in a $q-p$ grid. More specifically, the casualties are $q-p+1$ and the time is $\Theta(\frac{n}{d_1 \times \ldots \times d_p})$. In BVD-qG, the number of movements by LEA, EA and each SA is $O(n)$ and since there are at most q shadowing agents, the total number of movements until the BV is found is $O(qn)$ in the worst case. In PBVD-qG, the new value of "n" should be $\frac{q \times d_1 \times \ldots d_p \times n}{d_1 \times \ldots \times d_p}$ thus the number of movements should be $O(qn)$. $\square$

4.2.3 Tori

Informally, a torus is a mesh with "wrap-around" links that transform it into a regular graph. A torus of dimensions $d_1 \times d_2$ has $n = d_1 \times d_2$ nodes $v_{i,j} (1 \leq i \leq d_1, 1 \leq j \leq d_2)$ and each node has four neighbours which are $v_{i,j+1}, v_{i,j-1}, v_{i+1,j}, v_{i-1,j}$. The algorithm to parallelly decontaminate the BV in a torus, called PBVD-T, follows a strategy very similar to the one used for the 2-dimensional grid described in Section 4.2.1. There is only one main difference between the two strategies: in a 2-dimensional grid, all the agents move EAST in the exploration phase, while in a torus, because of the lack of borders, the spread of the BV might happen even if it reside in $v_{d_2,j}$ $(1 \leq j \leq d_2)$; therefore, we place another group of agents at $v_{i,0}$ $(i = 0, \ldots d_1 - 1)$ and these agents will act as a border and will stay here until the end of the exploration phase.

Initially, $2d_1$ agents are placed at $v_{0,i}$, $v_{1,i}$ $(i = 0, \ldots, d_1 - 1)$ (first round). If no agents are destroyed, then we place another d_1 agents in the first column: two agents at the top and the bottom as we do in the PBVD-2G. We then start the safe-exploration phase. If one of the agents is destroyed, assuming that the original BV resides in node $v_{0,j}$, $(1 \leq j \leq d_1)$, then all the clones of the BV are destroyed. If the BV resides in node $v_{1,j}$, $(1 \leq j \leq d_1)$, then the elimination phase begins. The movements of agents are the same as the ones of the agents in PBVD-2G. Note that in the exploration phase, only $2d_1 + 2$ agents actually move and d_1 agents simply stay in the first column to guard the nodes and ensure monotonicity. In the surrounding and elimination phase, the movements of the agents are also the same as those in PBVD-2G. Figure 4.12 shows the whole process of the PBVD-T when the BV resides in node (5,3) in a 5×6 torus. (one circle indicates the presence of one agent; two circles indicate two agents; three circle indicates three agents.)



(a) t_0 (b) t_1

(c) t_2 (d) t_3

(e) t_4 (f) t_5

(g) t_6 (h) t_7

Figure 4.12: Arrangement of agents PBVD-T

Let $\mathcal{T}$ be a torus of size $n = d_1 \times d_2 (d_1 > 2, d_2 > 2)$, with $d_1 = min\{d_1, d_2\}$.

Theorem 4. *The PBVD-T performs a decontamination of $\mathcal{T}$ using $3d_1 + 2$ agents (in the worst case, $3\sqrt{n} + 2$ agents) and at most 1 casualty, with at most $2n - \sqrt{n} + O(1)$ movements and at most time $\sqrt{n} + 11$.*

Proof. To complete the exploration and the elimination, we need $2(d_1 + 1)$ agents which has been alsready proven in Theorem 2, plus d_1 agents to guard the first column, so we need in total $3d_1 + 2$ agents (which is $3\sqrt{n} + 2$ in the worst case). The computation of casualties, number of movements, and time follow from Theorem 2. □

4.3 Summary

In this chapter, the BVD problems have been examined for two classes of interconnection networks: grids and tori. The parallel decontamination protocols for these two classes have been presented. Agents in our solution use only local information to execute the protocol. A summary of the results is shown in the table below, where q denotes the dimensions of the network, and n denote the numbers of nodes.

Chapter 5

Parallel Black Virus Decontamination in Chordal Rings

5.1 Introduction

In this Chapter, we discuss a parallel strategy for the BVD problem in chordal rings. In a chordal ring $C_n(d_1 = 1, d_2, ..., d_k)$, the nodes form a ring of size n and every node is directly connected to the nodes at distance d_i in the ring by additional links called chords. The link connecting two nodes is labeled by the distance that separates these two nodes on the ring (see Figure 5.1 for an example).

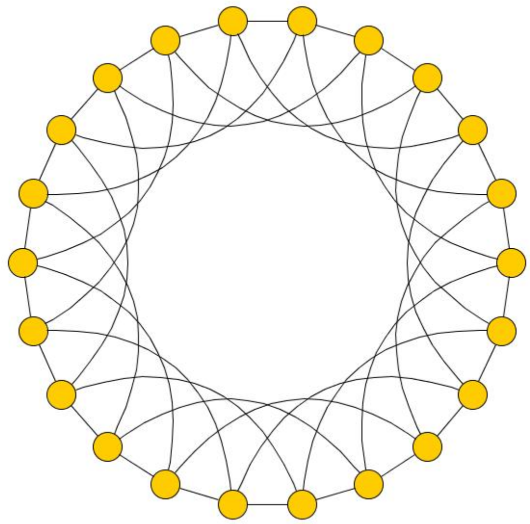


Figure 5.1: An example of chordal ring $C_n(1, 4)$

Agents and nodes are anonymous, but for the purpose of easier description, we refer to the nodes as x_0, x_1 .... Like in the case of meshes and tori treated in the previous Chapter, the agents are not allowed to communicate with each other unless they are in the same node so the protocol should enable agents in different nodes to move properly synchronized. That is, the route of every agent is different but they are served to explore the network; when a BV is triggered, other agents should bypass the new-formed BVs. We give simple but efficient solution to deal with the problem with acceptable cost. Our goal is to minimize the time to complete the whole decontamination process and at the same time the casualties. In order to do that, we propose a parallel strategy for decontaminating the chordal ring and this is the first attempt to deal with this issue in a parallel way.

5.2 Shadowed Exploration

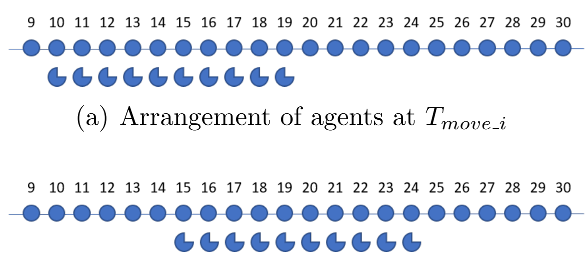
Initialization

The chordal ring is a complete symmetrical structure, so we can randomly choose a node x_0 as the start node. The initial setup consists of deploying three groups of agents. Initially, we place one agent in each of the first $2d$ nodes $x_0, x_1, \ldots, x_{2d-1}$. The agents residing in nodes from x_0 to x_{d-1} form the *shadowing group*, while the ones from x_d to x_{2d-1} form the *exploring group*. If the BV is within this window of nodes, then it is easily detected. We then assume that the first $2d$ nodes do not contain the BV, and we place d additional agents at nodes $x_0, x_1, \ldots, x_{d-1}$ (*guarding group*). Only the shadowing and exploring groups move to explore the graph. The ones in the guarding group remain dormant for now, guarding the nodes to guarantee monotonicity.

Route of the agent in exploring phase

The exploration proceeds in synchronized rounds composed each by one movement step, when selected groups of agents move to proceed with the exploration, and three steps for synchronization purposes. We call these different steps *move step* and *notification steps*. A round T_i is composed by four time units, one for the move step, and 3 for the notification steps: $T_1 = T_{move\_1}, T_{noti\_1^{(1)}}, T_{noti\_1^{(2)}}, T_{noti\_1^{(3)}}, T_2 = T_{move\_2}, T_{noti\_2^{(1)}}, T_{noti2^{(2)}}, T_{noti\_2^{(3)}}, \ldots$

The agents that move during a move step T_{move_i}, always do so along their longest chord d_k. That is, agents move along d_k in steps $T = 1 + 4t$ ($t \in \mathbb{N}$). An example of how agents move in chordal ring $C_n(1, 2, 4, 5)$ at time T_{move_i} in the exploring phase is shown in Fig. 5.2.



(a) Arrangement of agents at T_{move_i}

(b) Arrangement of agents at T_{move_i+1}

Figure 5.2: Arrangement of agents when moving

Synchronization: Three Jump Notifying Technique In the sequential strategy [1], two agents explore the graph (exploring agent and leader agent) using "cautious walk". That is, the Exploring Agent moves to the next node in its route, and if the node is safe, it moves back to the Leader Explorer Agent, and then they move forward to that safe node together. If instead the node contains a BV, the Leader Explorer Agent becomes aware of that because a BV arrives through that link instead of the Explorer Agent. However, in our strategy, we employ $2d$ agents in the exploring phase but we do not use the "catious walk". We then have to guarantee that all agents find out whether or not the BV has been found in the current round. In fact, if they are not properly informed, when one agent is destroyed by the BV, in their next step some of them (the risky agents RAs) may be destroyed by the newly formed BVs. In order to avoid these potential casualties, we propose the $Three\,Jump\,Notifying\,Technique$ to properly notify the agents that otherwise would move to the newly formed BVs in the next round.

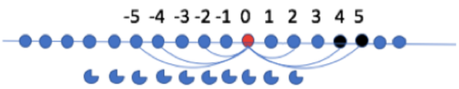
The idea is the following: when/if a node receives a clone and becomes aware of the presence of the BV, it becomes a $Notification\,Agent$ (NA). The NA's role is to make the risky agents aware of the presence of the BV in an efficient way. They will do so in parallel, each following a special route of length 3. More precisely, let the BV be at node x_0 (refer to Figure 5.3), the $Notification\,Agent$ located at node x_{n-d_i} will follow the following route:

$$x_{n-d_i} \xrightarrow{move\,along\,chord\,d_i} x_0 \xrightarrow{move\,along\,chord\,d_k} x_{n-d_k} \xrightarrow{move\,along\,chord\,d_i} x_{n-d_k+d_i}.$$

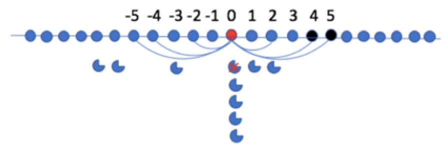
In this case, the notifying route of the NA whose coordinate is x_{n-d_k} is $x_{n-d_k}{\rightarrow}x_0{\rightarrow}x_{-d_k}{\rightarrow}x_0$.

Note that, with the $Three\,Jump\,Notifying$ technique, we only inform the RAs, that is the agents that will be destroyed by the BV in the next step (the ones residing in nodes $x_{n+d_i-d_k}$, $(i = 1, \ldots, d_{k-1})$); however, agents (if any) residing in node $x_{n+d_i-2d_k}$, $(i = 1, \ldots, d_{k-1})$ would still be destroyed by the BVs in their next two moves. So they should be properly informed as well. This could be easily performed as follows: when one agent $A1$ moves to a node where there is an agent $A2$ knowing the position of the original BV, $A1$ becomes informed and directly moves along the longest chord to its own position. In this way, all the agents that might be destroyed by the BV will be properly informed.

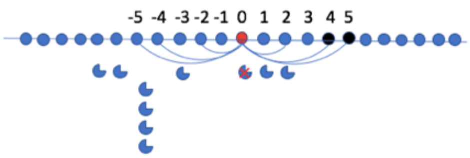
We will make some modification of this agents route in the *Surrounding and Elimination*, but for now let us assume it follows the route above. The whole process of *Three Jump Notifying* technique in chordal ring $C_n(1, 2, 4, 5)$ is shown in Figure 5.3.



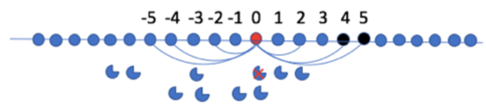
(a) Arrangement of agents at T_{move_i}



(b) Arrangement of agents at $T_{noti\_i^{(1)}}$



(c) Arrangement of agents at $T_{noti\_i^{(2)}}$



(d) Arrangement of agents at $T_{noti\_i^{(3)}}$

Figure 5.3: The whole process of the *Three Jump Notifying* technique in chordal ring $C_n(1, 2, 4, 5)$

If the original BV is residing in the red node, then once an agent moves to it, the agent and the BV are destroyed but the clones of the BV spread to all its neighbours. According to our technique, agents residing in nodes x_{-4}, x_{-3}, x_{-1}, x_0 are the ones to be notified; agents residing in nodes x_{-5}, x_{-4}, x_{-2}, x_{-1} are the NAs. The routes for agents residing in nodes x_{-5}, x_{-4}, x_{-2}, x_{-1} are $x_{-5}{\rightarrow}x_0{\rightarrow}x_{-5}{\rightarrow}x_0$; $x_{-4}{\rightarrow}x_0{\rightarrow}x_{-5}{\rightarrow}x_{-1}$; $x_{-2}{\rightarrow}x_0{\rightarrow}x_{-5}{\rightarrow}x_{-3}$; $x_{-1}{\rightarrow}x_0{\rightarrow}x_{-5}{\rightarrow}x_{-4}$ respectively.

Safe Exploring with Three Jump Notifying technique

After the initialization, the exploring and shadowing agents move following the longest chord. The subsequent three time units are either used for the notification phase (if one of the agents is destroyed at time T_{move_i}), or they are spent simply by waiting before the next move. If executing the *Three Jump Notifying* technique, the NAs move back to where they were before the notification. For example, in the example in *Three Jump Notifying* technique, NA residing in node -1 moves back to node x_{-4} following the reverse route in the notifying phase which is $x_{-1} \rightarrow x_{-5} \rightarrow x_0 \rightarrow x_{-4}$.

5.3 Surrounding and Elimination

When a BV is found, the Three Jump Notifying technique guarantees that the risky agents are now aware of the presence of the BV. Other agents, however, might not have received the notification and might proceed to the next round without such knowledge; we call these agents $KeepMoving$ agents.

In this section, we introduce the process of eliminating the BVs after the original BV is triggered. For the purpose of saving the number of agents, we prefer to chase the $Keep\,Moving$ agents, but it is not necessary to complete the process especially when you care most about the execution time; In that case, we may instead carry enough agents and proceed to the $Surrounding\,and\,Elimination$ phase immediately. The number of agents that should be carried in order to successfully proceed the $Surrounding\,and\,Elimination$ will be discussed later. We now describe how to chase the $KeepMoving$ agents.

5.3.1 Notifying Moving Agents

Overview of the NotifyingMovingAgents phase

When the $Shadowed\,Exploring$ ends, it is possible that some of the agents are not informed and do not realize the existence of the BV, so they keep moving following the routes of the $Shadowed\,Exploring$ phase but it is obvious that they would not encounter any BV. In order to reduce waste, we employ the agent who receives the clone from chord d_k ($Coordinator\,Agent$) to notify the other $Keep\,Moving$ agents to move back to the positions they occupied before the BV was triggered.

The Process of the Notifying Phase of the Coordination Agent

To do that we employ one of the agents as a *Coordinator Agent*(CA). The CA follows a specific path that will guarantee to meet all the *Keep Moving* within a certain amount of rounds.

The general outline of the technique is the following:

1. the CA is chosen to be the agent who receives a BV clone from its longest chord.

2. following three moving rules (the rules of moving are described later) the CA moves to occupy a node as the starting point for chasing and sets a notification window $[x_y, x_z]$ (the range computation is described later) based on its own coordinate.

3. the CA waits an appropriate amount of time to allow the agents that are still moving to reach this window of nodes

4. the CA now moves in synchronization with the movement of the agents and follows a specific paths. More precisely, while the moving agents proceed as usual with one move and 3 waiting steps, the CA takes its longest chord in correspondence of an agents movement, and 3 consecutive nodes in correspondence of the waiting steps of the moving agents. In doing so, with $O(d_k)$ moves, the CA is guaranteed to have encountered all of them.

5. when a moving agent encounters the CA, it stops and waits for a second agent that will arrive at the next round. When both agents are there, they go back to their original positions to start the surrounding phase.

For convenience, we consider the chordal ring as arranged in rows of size d_k where the last node of a row is connected to the first node of the following row and the last node is connected to the first. Depending on the size of the chordal ring, the last row could be incomplete. So in this matrix, moving down a column corresponding to using the longest chord d_k.(see Figure 5.4).

0	1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43
44	45	46	47	48	49	50	51	52	53	54

Figure 5.4: Viewing a chordal ring whose longest chord is 11 as a matrix

In the following step, we use a concept called "Notification Window". In a chordal ring $C_n(1, d_2, \ldots, d_k)$, given a coordinate x, the $Notification\,Window$ $[x_i, x_j]$ contains a set of consecutive nodes where x_y is the $Beginning\,Flag$ and x_z is the $End\,Flag$. When we mention "marking a flag", it does not mean that the agent has to move to the node to do that but only needs to remember the positions of the two flags in its memory. The relations between x, x_y, x_z is defined as follows:

- x_y is the biggest number smaller than or equal to x_i and such that x_y mod d_k =0;
- x_z is the smallest number s bigger than x_i and that x_z mod $d_k = d_{k-1}$.

For example, in the matrix of Figure 5.4, given a node, the $Beginning\,Flag$ of its "Notification Window" is the first node of its row while the $End\,Flag$ is the last node of its row.

The selection of the CA is simple: when an agent receives a BV clone from its longest chord, it realizes that it is chosen as the CA. More specifically, if the coordinate of the original BV is x_i, then the coordinate of the CA would be x_{i-d_k}. After being selected as the CA, the CA should move to a node as the starting point for chasing. In the matrix, if the original BV reside in row i, then the destination of the CA (the starting point for chasing) should be in any node in row $i+2$ to avoid being destroyed by the clones when chasing the "Keep Moving" agents. In this case, the $Notification\,Window$ set by the CA would be from $x_{d_k \times (i+2)}$ to $x_{d_k \times (i+2)+d_k-1}$.

Supposing the coordinate of the original BV is x_i, the coordinates of the positions where the clones spread are: x_{i-d_k} (which is the original coordinate of the CA), $x_{i-d_{k-1}}$, $x_{i-d_{k-2}}$,

$\ldots$, x_{i-1}, x_{i+1}, x_{i+d_2}, $\ldots$, $x_{i+d_{k-1}}$, x_{i+d_k}. There are three different situations to consider to describe the behavior of the CS. Now we describe the three scenarios:

- Scenario 1: The last agent of the *Exploring Group* is destroyed by the BV and the positions of the clones satisfy: $x_{i-d_{k-1}+d_k} = x_{i+1}$, $x_{i-d_{k-2}+d_k} = x_{i+d_2}$, $\ldots$, $x_{i-1+d_k} = x_{i+d_{k-1}}$.
- Scenario 2: The last agent of the *Exploring Group* is destroyed by the BV and the at least pair of the positions of the clones does not satisfy: $x_{i-d_{k-1}+d_k} = x_{i+1}$, $x_{i-d_{k-2}+d_k} = x_{i+d_2}$, $\ldots$, $x_{i-1+d_k} = x_{i+d_{k-1}}$.
- Scenario 3: One of the agents in the *Exploring Group* except the last agent is destroyed by the BV.

In scenario 1, the CA needs to move for 5 steps to reach its destination while in the other two scenarios, it only needs to move for 4 steps to arrive the destination. Now we describe the route for each scenario.

- For CA in scenario 1: Let us denote by x_i the coordinate of the node in the $Notification\,Window$ set by the coordinate of the original BV which does not receive any clone and his left neighbour receives a clone (the coordinate of which is x_{i-1}). The CA first moves to the original BV, then to node x_{i-1}, finally to x_i. At this point, it only needs to move along chord d_k twice to reach its destination.
- For CA in scenario 2: There is at least one pair of positions of the clones that does not satisfy the equations, so there should be one node (say, x_i) who receives a clone from the original BV but node x_{i+d_k} is empty. The route now for the CA is first to move to the original BV, then to node x_i, and then along the chord d_k twice to reach its destination.
- For CA in scenario 3: The CA here simply needs to move one step to its right neighbour and move along the chord d_k three times to reach its destination.

The $Keep\,Moving$ agents reach the CA's $NotificationWindow$ 8 unit of time after the BV is triggered. Because it takes at most 6 units of time for the CA to reach its destination, the CA needs to wait for the $Keep\,Moving$ agents before beginning the chasing. The CA simply counts the time it costs to reach its destination (say, t), and computes the time it needs to wait, which is $8 - t$. The chasing phase starts from time $T_{move\_i+2^{(1)}}$. The *KeepMoving* agents proceed with one move and three waiting steps, while the CA takes its longest chord in correspondence of an agent's move and three consecutive nodes in correspondence of the waiting steps of the *KeepMoving* agents. Every time when the CA moves along a chord d_k, it resets the "Notification Window" as below:

$New\,Beginning\,Flag = Old\,Beginning\,Flag + d_k$

$New\,End\,Flag = Old\,End\,Flag + d_k$

When the CA moves along three consecutive nodes, it notifies the agents it encounters (if any) to go back. To ensure that the CA and the *KeepMoving* agents are in the same Notification Window, when the CA, during its movement, realizes that it just passes the $End\,Flag$, the CA moves along the longest chord counter-clockwise to the node marked $Beginning\,Flag$ and continues to move along three consecutive nodes to notify the agents it encounters. The chasing phase terminates when the CA arrives at a node x_y that satisfies the condition $x_y = x_z + t \times d_k$ ($t \in \mathbb{N}$) (the relative starting point).

Example: The route of the CA in the chasing phase is shown in the example of Figure 5.5 for a chordal ring $C_n(1, 2, 7, 11)$.

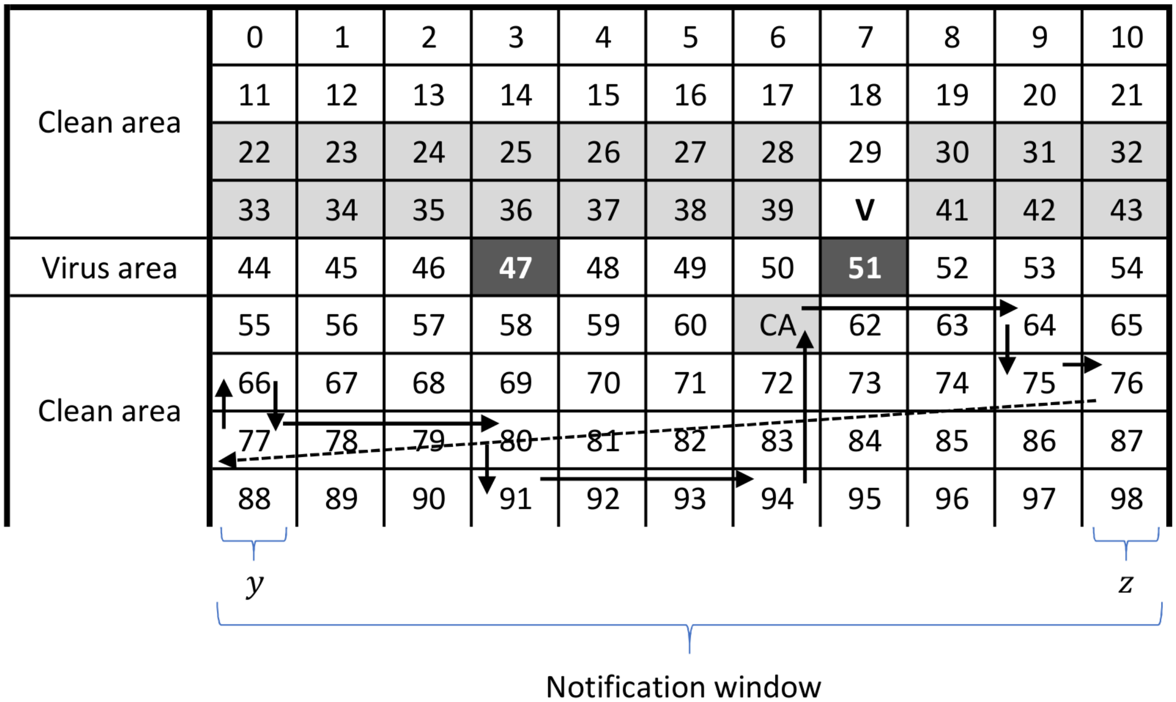


Figure 5.5: The route of the CA in the chasing phase in a chordal ring C(1, 7, 11)

In this example, node x_{61} is the starting point of the chasing phase and at this time, the Notification Window is $[x_{55}, x_{65}]$. The CA moves along three consecutive nodes x_{62}, x_{63} and x_{64} while the *KeepMoving* agents are waiting and then move along the longest chord to node x_{75}. At this time, the CA updates its Notification Window to $[x_{66}, x_{76}]$. After that, it continues to move along three consecutive nodes, but when it arrives atx_{77}, it realizes that it just passes the *End Flag* which is x_{76}, so it moves along the longest chord counter-clockwise to the node marked *Beginning Flag* which is x_{66}. After that, it moves along the longest chord with the *KeepMoving* agents to x_{77} and again updates its Notification Window, then moves along the nodes to notify the agents.

Let us assume that the time when the original BV is triggered is T_{move_i} , then the CA remember s this time and informs the agents it encounters of it. The agent $A1$ that encounters the CA remember the encounter time $(T_{noti\_y^{(a)}}\ a \in (1, 2, 3))$ and stops moving until the next T_{move} (T_{move_y+1}) when it will meet another agent $A2$. Then $A1$ moves along

d_k counter-clockwise for $y+1-i$ time units while $A2$ moves for $y+2-i$ time units. When arriving at its relative starting point at time T_{move_a}, the CA knows that it has finished the chasing phase, and it moves along chord d_k to its position when the original BV is triggered.

We now describe how the agents and the CA move in coordination in the example of $C_n(1,2,7,11)$ (see Figure 5.6).

0	1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	V	41	42	43
44	45	46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63	64	65

Figure 5.6: Arrangement of agent at T_{move_2} when the BV is triggered

Yellow nodes are connected to the original BV but guarded by agents while grey nodes are the new formed BVs. The node marked V is the original BV, which is now clean. The agent residing in node x_{29} receives a clone from chord d_k so it knows it is the CA. During the notifying time, agents residing in nodes x_{33}, x_{38}, x_{39} notify agents residing in nodes x_{36}, x_{31}, x_{30} respectively following the $Three\,Jump\,Notifying\,Technique$, while the CA moves to x_{28}, x_{39}, x_{50} and finally x_{61} following the route in scenario 3.

0	1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	V	41	42	43
44	45	46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63	64	65
66	67	68	69	70	71	72	73	74	75	76

Figure 5.7: Agents' roles after $Three\,Jump\,Notifying\,Technique$. (for convenience, we denote the CA by a red spot, more specifically, node x_{50} is where CA resides)

Agents in purple nodes would be notified at time T_{move_3} and move back. Agents in light green nodes are the $Keep\,Moving$ agents, while agents in dark green nodes are informed to stop during the $Three\,Jump\,Notifying\,Technique$. In the meantime, the CA moves to node x_{28}, x_{39}, x_{50}, and finally x_{61}. It is obvious that the CA can reach its destination before time T_{move_4}, so it waits until $T_{noti\_4^{(1)}}$ to start its notifying phase.

0	1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	V	41	42	43
44	45	46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63	64	65
66	67	68	69	70	71	72	73	74	75	76

Figure 5.8: Arrangement of agent at T_{move_3}. The CA has arrived its destination

0	1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	V	41	42	43
44	45	46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63	64	65
66	67	68	69	70	71	72	73	74	75	76

Figure 5.9: Arrangement of agents at T_{move_4}. The CA starts its chasing phase

In the chasing phase, CA starts to notify other $Keep\,Moving$ agents. First, it computes the $Notification\,Window$, which is from node x_{55} to node x_{65}. It moves to node x_{62} at time $T_{noti\_4^{(1)}}$, to node x_{63} at time $T_{noti\_4^{(2)}}$, to node x_{64} at time $T_{noti\_4^{(3)}}$ and to node x_{75} at time T_{move_5}.

0	1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	V	41	42	43
44	45	46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63	64	65
66	67	68	69	70	71	72	73	74	75	76
77	78	79	80	81	82	83	84	85	86	87

Figure 5.10: Arrangement of agents at T_{move_5}.

After that, CA moves to node x_{76} at $T_{noti\_5^{(1)}}$. We can see that it encounters the agent residing in node x_{76}, so CA informs node x_{76} about the time $T_{trigger}$, which is T_{move_2}. The agent residing in node x_{76} remembers the time T_{noti_now}, which is $T_{noti\_5^{(1)}}$, and waits additional time T_{move} to inform agent ($Following\,Agent$) that resides in node x_{65} to move to node x_{76} after waiting T_{move} time units. After encountering its $Following\,Agent$, the CA informs it to move back along chord d_k for $T_{move\_now} - T_{trigger} + 1$ time units (which correspond to $T_{move\_5} - T_{move\_2} + 1$ time units), while it moves for $T_{move\_5} - T_{move\_2}$ time units.

At time $T_{noti\_5^{(2)}}$, when the CA arrives at node x_{77}, it knows that it just pass its $Ending\,Flag$ so it moves along the longest chord anticlockwise to its $Beginning\,Flag$ (node x_{66} at $T_{noti\_5^{(3)}}$).

0	1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	V	41	42	43
44	45	46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63	64	65
66	67	68	69	70	71	72	73	74	75	76
77	78	79	80	81	82	83	84	85	86	87

Figure 5.11: Arrangement of agents at $T_{move\_5^{(3)}}$.

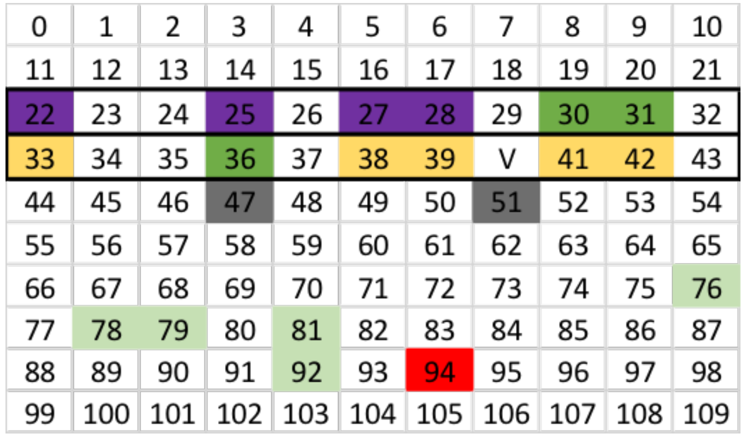


Figure 5.12: Arrangement of agents at $T_{move\_7^{(3)}}$.

The CA moves back to its relative starting point (node x_{94}) at time $T_{noti\_7^{(3)}}$ and it now knows that it has finished the chasing phase; so, it moves back to its original starting point, node x_{29}.

5.3.2 Overview of the Elimination

After all the agents move back to where they are when the BV is triggered, they start the *Surrounding and Elimination* phase.

The BVs can be destroyed sequentially, which is simple, but might be time consuming in some cases. In this way, because all the agents are aware of the positions of the new-formed BVs, for each BV, at most $d-1$ agents are sent to surround it and one agent is sent to destroy. This elimination strategy has been used in [1].

An alternative strategy, which allows a faster decontamination at the expense of a larger number of agents, is to eliminate the BVs concurrently. First we need to guard all the neighbouring nodes of all the newly formed BVs. In order to avoid collision and efficiently leverage the agents, we allocate different Destination Tables to all the agents to inform them where should they move in different situations (e.g., when the first agent in the exploring team is destroyed, then every agent except the first agent have a distinct destination, when the second agent is destroyed, then every agent except that agent destroyed have again a distinct destination). More specifically, for a Chordal Ring with degree $2k$, every agent in carries a $DestinationTable$ with $k-1$ destinations. If we need more agents, then we give their $DestinationTable$ to the last agent in the shadowing group. When the elimination begins, the agent clones a sufficient number of agents and give them the $DestinationTable$. Before moving to its destination, an agent computes the shortest route from its own position to its destination using Dijkstra Algorithm. There are two kinds of agent in the Elimination phase: *surrounding agents* that are responsible for guarding the neighbouring nodes of the BVs and *destroying agents* that move to the BVs after all the neighbouring nodes are guarded. We want the BVs to be destroyed simultaneously, so it is important that the destroying agents move to the BVs at the same time and only after all the neighboring nodes are guarded by agents. In fact, if the destroying agents know the longest time $t_{longest}$ to move to the destination taken by all the agents (including the destroying agents and the surrounding agents), then they can move to the last node prior to the destination and wait until time $t_{longest}$ to move to the BVs simultaneously. So, in the $DestinationTables$ for the destroying agents, we also add the time $t_{longest}$. We now describe how to compute the shortest routes and how to design the $DestinationTables$. Note that we design $DestinationTables$ for all the agents and allocate them to the agents before the exploring phase begins.

5.3.3 Destination Table and Elimination

Suppose that there are some BVs and agents in the chordal ring, it is obvious that the BV nodes are in the clockwise side of the agents. In order to use Dijkstra, first we need to map the chordal ring with BVs into a sub-graph where we would run the Dijkstra algorithm to find out the shortest route from every node to its destination. More precisely, the sub-graph (i) includes only the nodes from the node containing the first agent clock-wise to the node which is d_k away from the last BV node (ii) does not include the links incident to the BV nodes. Here is an example how we built the graph for running Dijkstra Algorithm. Below we show the situation when the third agent in the exploring group is destroyed by the BV (see Figure 5.13). Only the chords of the original BV node are shown.

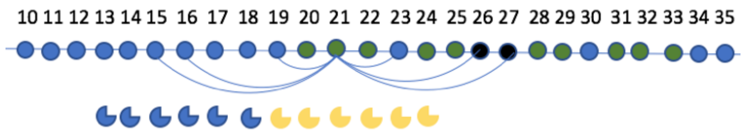


Figure 5.13: Situation when the third agent in the exploring group is destroyed

The black node is the BV node while the green nodes need to be guarded. In this case, we need 12 agents (10 surrounding agents and 2 destroying agents). We add nodes from 13 to 33 with their chords within this area and delete chords connected with the BV nodes to get the graph where we use Dijkstra Algorithm. Below is the graph we build. (see Figure 5.14). For convenience, we show all the nodes we included and the chords we delete.

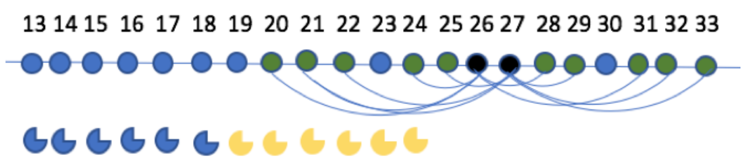


Figure 5.14: The graph we build for Dijkstra Algorithm

Using the graph and Dijkstra Algorithm, we compute the routes from every agent to every node. Then we use enumeration to choose an allocation of every agents's destination considering that:

1) the maximum length of the route should be minimum.

2) after the allocation, in every needed position there should be exactly one agent.

Every agent in the exploring group should hold a *DestinationTable* of $d-1$ parts (Part i records its moving information when (anticlockwise) agent i in the exploring group is destroyed, except itself) while every agent in the shadowing group should hold a $DestinationTable$ of d parts. Besides, every part contains 1 items (for surrounding agents) or 2 items (for destroying agents). After we get the optimal allocation, we can know the destination of every agent for each destroyed agent, and we can also know whether it is a surrounding agent or a destroying agent in the situation when agent i is destroyed.

For each agent in the chordal ring, each item in Part i of its $DestinationTable$ is arranged as below.

1) The first item records the agent's destination

2) The second item records the longest time $t_{longest}$ among all agents.

After one of the agents is destroyed, the agents can check their $DestinationTable$ to get the information of their destination. Then using Dijkstra Algorithm they can compute the shortest routes separately and start to move. The destroying agents should move to the last node prior to the destination and wait until time $t_{longest}$ to move to the BVs together.

Time cost analysis and Comparisons

We only consider the situation when n (the number of nodes of the chordal ring) is much larger than d_k, and since the time cost in the elimination phase is $O(1)$, we only compute the time cost in the exploring phase. Finally, we compute the TWT(calculated by multiplying the size of the team and the time cost by the solution) of both protocols to present a more fair comparison.

Let us assume that the total number of moves is M, then the worst case costing the most time is when the BV is located at any nodes within the range from x_{n-d_k+1} to x_{n-1}. In this case, it cost $M = \lfloor 2n - 2d_k \rfloor$ moves and $4\frac{M}{2d_k}$ units of time which is $4 \times \left\lfloor \frac{n}{d_k} - 1 \right\rfloor$ to finish the exploring phase. In [1], the author gives the number of move in three cases.

- 1) In double loops the upper bound of moves is $4n - 7$.

- 2) In the triple loops, two classes of chordal rings are discussed: $C_n(1, p, k)$ and $C_n(1, d_k - 1, d_k)$. In the first case, the number of moves needed is $5n - 6d_k + 22$ while in the second case, a maximum of $5n - 7d_k + 22$ moves are needed.

- 3) In the consecutive-chordal rings, a maximum of $(d_k + 2)n - 2d_k - 3$ moves are needed.

Since in the sequential strategy, agents do not need to wait, the time cost is equal to the number of moves. And it is obvious that our protocol is much faster than the sequential strategy. But since we use much more agents, in order to gain a fair comparison, we compute the TWT measure for both protocol.

In the exploring phase, we use $2d_k$ agents, so the TWT of our protocol is $8n - 8d_k$. The exploring phase of the sequential strategy needs at least 2 agents to explore and some other shadow agents to guard the explored nodes; the number of shadows depends on the structure of the chordal ring, so for now we ignore them. Now we compute TWT of the sequential strategy.

- 1)In double loops the upper bound of TWT on $8n - 14$

- 2)The TWT in chordal rings $C_n(1, p, d_k)$ is $10n - 6d_k + 44$ and in chordal rings $C_n(1, d_k - 1, d_k)$ is $10n - 14d_k + 44$.

- 3)The TWT in consecutive-chordal rings is $2(d_k + 2)n - 4d_k - 6$.

It is obvious that when $d_k \geq 2$, our protocol is faster in the first case; when $d_k \leq \frac{1}{3}n + 7$,our protocol is faster in the second case (both $C_n(1, p, k)$ and $C_n(1, d_k - 1, d_k)$); when $d > 2 - \frac{1}{n+2}$, our protocol is faster in the third case.

I would put tables with results

Casualty Analysis

Casualty is the number of agents destroyed by the BV. In chordal ring $C_n(1, d_2, \ldots, d_k)$, the worst case is when the first agent in the exploring group is destroyed by a BV and its clones spread to all its neighbouring nodes. The casualties in this case are $d_k + 1$ because another d_k nodes are guarded by agents. On the other hand, in sequential case, the casualties are $2d_k$. So in terms of casualty, our protocol is better than the sequential strategy.

I would put another small table with the comparison

Chapter 6

Parallel Black Virus Decontamination in Arbitrary Graph

6.1 Introduction

In [5], Cai proposes two exploration strategies: Greedy Exploration and Threshold Exploration, both spread optimal and total number of agents asymptotical optimal. Since these strategies are sequential, they are time consuming($O(\Delta n^2)$). In order to explore the graph in parallel, we propose two different strategies:

(1) Flood Strategy

(2) Castle-First Strategy

The general idea of the Flood Strategy is simple: supposing that an agent resides in node v, and it has m neighbours to be explored, then it simply clones m agents and send them to its neighbours. In the Castle-First Strategy, we build "castles" which are structures composed by a single node or by a combination of several nodes with special properties (rules are introduced later); the exploration phase can be viewed as the combination of several smaller scale explorations in the graph and it begins with the location of one of the exploring group and ends with one of the unexplored castles. We will see that after all the castles are explored, all the nodes in the graph are explored. The general exploring algorithm for these two strategies is based on the one described in Chapter 3 which consists of performing a *Shadowed Exploration* phase to locate the BV, followed by a *Surrounding and Elimination* phase to eliminate the cloned BVs.

Strategy *Flood* is time optimal with the cost of a great number of agents, while strategy *Castle-First* comes to a compromise between *Flood* and the sequential strategies: it employ much less agents than *Flood* and it costs much less than the sequential strategies in terms of time.

In the arbitrary graph, we assume that the original BV and its clones do not disconnect the graph.

6.2 The Flood Strategy

Initialization. In this strategy, we assume that all the agents are endowed with 3-hop visibility. For this strategy, in contrast with the sequential strategy, agents do not need memory to remember the routes they pass by. Finally, in this strategy, we use an important ability of agents which is "cloning". As we introduced in Chapter 2, cloning means that an agent is endowed with the capacity to generate one or more agent identical to itself.

We use the Dijkstra Algorithm to compute the shortest route from the homebase to every node and write the route step on the white board on each node, so for each node, there should be a number (Shortest Route Number) recording the number of steps of the shortest route from the homebase to it. Let us denote by v_{SRN} the Shortest Route Number of node v, and nodes $v_1, \ldots, v_i$ are neighbours of node v assuming that v has i neighbours. Then edges connecting node v and its neighbours are $e(v, v_1), \ldots, e(v, v_i)$. We write the v_{SRN} in correspondence to the end port of these edges (the end connecting to its neighbours), so when an agent resides in any node from v_1 to v_i, it can see the Shortest Route Number of node v because of the local visibility (see, for example, Figure 6.1).

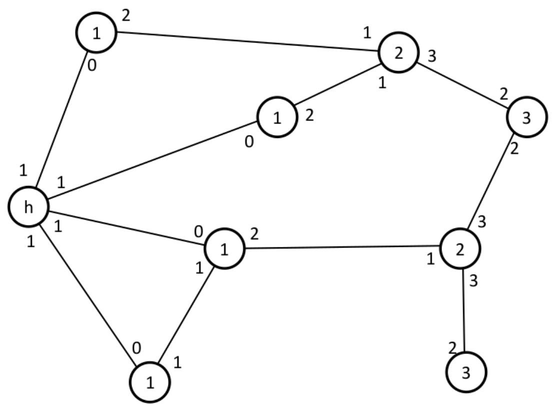


Figure 6.1: Initialization of the graph

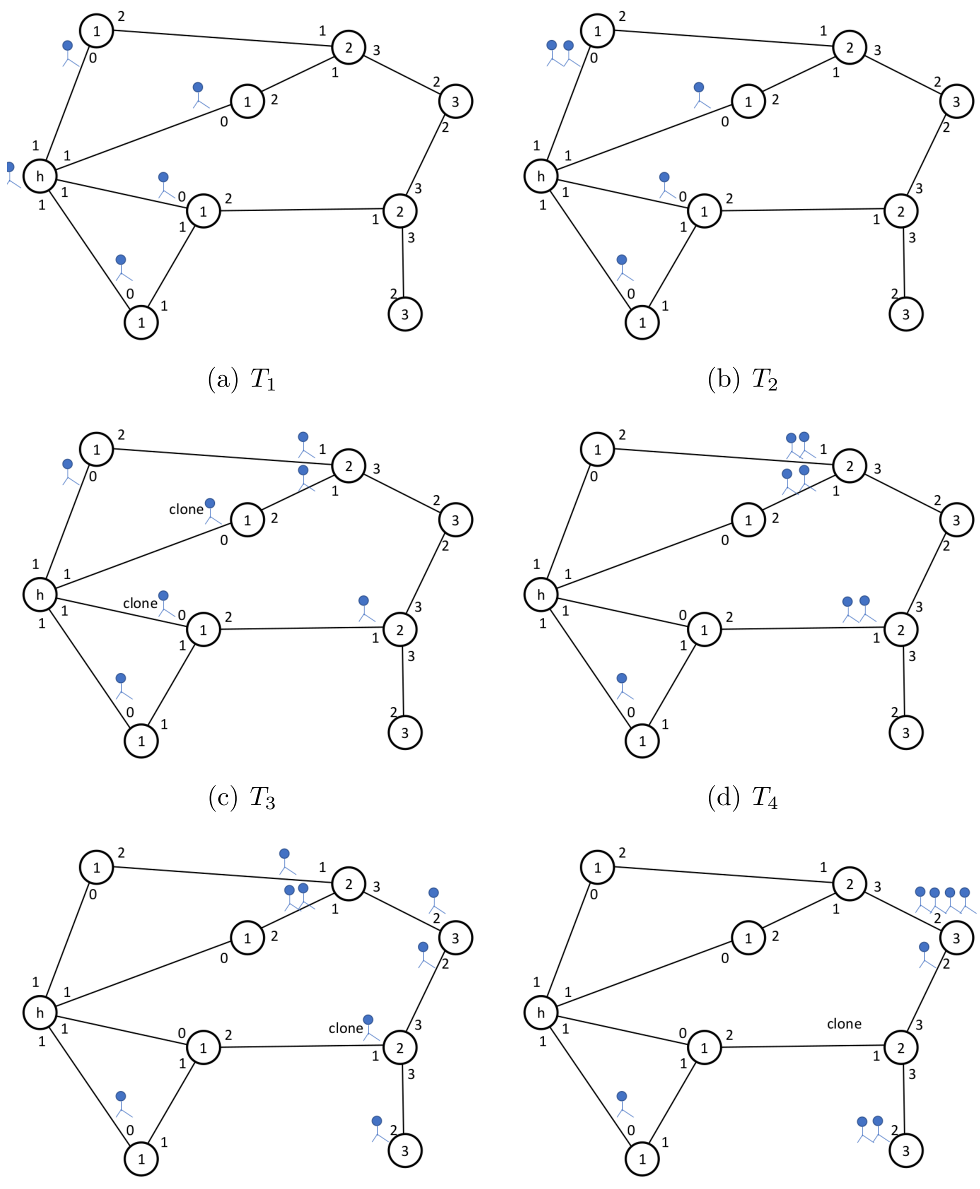
Exploration Phase All the agents in this strategy follow the same rules in the exploration phase. Let us denote by SNR the Shortest Route Number.

1. Agents can only move from node with lower SNR to node with higher SNR.

2. Let m agents reside in node v with SNR of v equal to a, and let the next destination(s) (node v's neighbours with higher SNR than v) be $\{v_1, \ldots, v_k\}$ (such neighbours always exist).

   - if $m < k+1$ (the number of agents is not sufficient), then one of the agents residing in node v clones $k+1-m$ agents. After that, k agents of the agents residing in node v move to the k destinations while the remaining agents stay in node v to guard v at T_i. If one of the agents is destroyed, the Elimination phase begins; if none of the agents are destroyed, the remaining $m-k$ agents move to any of the k destinations at time T_{i+1}. - If an agent resides in a node without any neighbour whose SNR is equal to $a+1$, then it simply stays there. Since all the action of agents happen after they meet each other at the same node, they can communicate with each other and make sure that all their routes do not conflict.

An example of how the agents move is showed in Figure 6.2.



(a) T_1 (b) T_2

(c) T_3 (d) T_4

(e) T_5 (f) T_6

Figure 6.2: An example of how the agents move in the Flood Strategy

Elimination Phase. Let us assume that the node where the original BV resides is v with SNR equal to a and that it is triggered at time T_i. Then at this time, the clones spread to all neighbours of node v with SNR equal to $a+1$ and survive while leaving node v clean (no agent and no BV) and let us denote by v_{BV}s all these BV nodes.

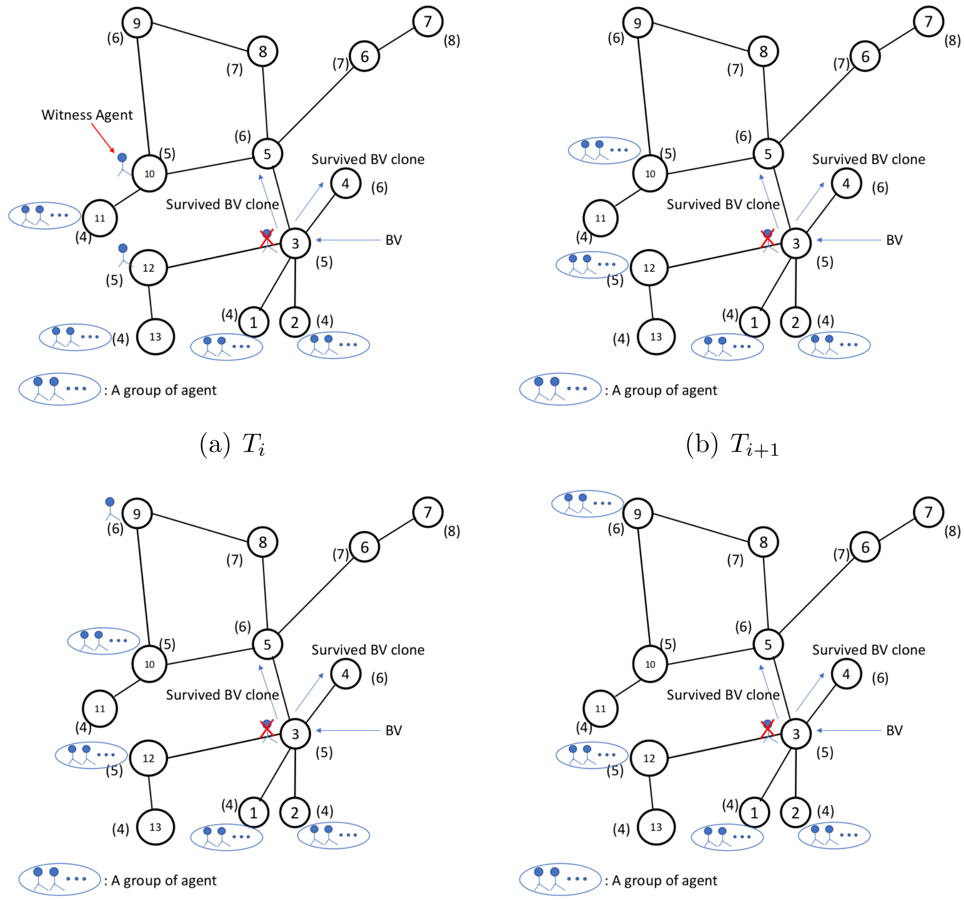
We now describe how agents residing in different positions move in the Elimination phase.

1. Rule 1: For convenience, we call agents residing in the v_{BV}s' neighbours with SNR equal to a the *Witness Agents*. Note that the Witness Agents can easily realize whether or not node v is the place where the original BV resides. In fact, since they have 3-hop visibility, if they see that one of their "2-distance" neighbours (say node v') does not contains an agent but some neighbours of node v' contain agents, they know that the original BV resides in node v. If the Witness Agents realize the existence of BV at time T_i, then they simply stop cloning and moving to the new formed BV nodes. Note that if the Witness Agents (say it resides at some node u) have some higher SNR neighbours except for the new formed BVs, they should still explore these nodes following the rules in the exploration phase but leaving an agent in node u to guard it.

2. Rule 2: For all of the agents residing in node v's neighbours with SNR equal to $a-1$ (say there are k such agents), they receive clones and understand the location of the BV, and should move to v at time T_{i+2}. Let us denote by p the number of v's neighbours with SNR equal to $a+1$, then if $k < p+1$, one of the agents residing in node v should clone another $p+1-k$ agents and then p agents move to v's neighbours with SNR equal to $a+1$ at T_{i+4}.

Note that in our assumptions, the original BV and its clones do not disconnect the graph, so there should be alternative routes from the homebase to the new BV nodes, even excluding the original BV node. So, when the agents are sent to v's neighbours with SNR equal to $a+1$ at time T_{i+4}, some other agents move to these nodes's higher SNR neighbours at the same time, which ensure that all the neighbours of the new formed BVs are guarded when they are triggered.

See Figure 6.3. For convenience of description, we give every node in the graph a distinct ID from 1 to 13. As shown in the picture, when the BV is triggered, the clones spread to all neighbours of node 3; node 4 and 5 become new BV nodes while nodes 1, 2 and 12 destroy a clone each and understand that the original BV resides in node 3.



(a) T_i (b) T_{i+1}

(c) T_{i+2} (d) T_{i+3}

Figure 6.3: A possible situation when the BV is detected. The Ids of the nodes are written inside the nodes, their SNR value is indicated in parenthesis.

In Figure 6.3, an agent residing in node 10 is a Witness Agent and, because of the 3-hop visibility, it can "see" that there is no agent residing in node 3 but there are agents residing in node 3's neighbours (node 1 and node 2). In this way, it knows that the original BV resides in node 3, therefore, it stops moving and cloning towards the BV but, since it has still a higher SNR neighbour (node 9) which does not contain a BV, it continues the exploration in that direction following the rule in the exploration phase. That is, it sends an agent to node 9, then all the agents except one move to node 9. After that, one agent is sent to node 8.

Theorem 5. *Let h be the diameter of the graph. The Flood Strategy performs the exploration phase in $2h$ unit of time.*

Proof. According to the rules in the exploration phase, the agents keep exploring when they have no more higher SNR neighbours, which means the exploration phase ends when the node(s) with the highest SNR is explored and this SNR should be the diameter of the graph. Since exploring nodes with the same SNR requires 2 units of time, the exploration phase needs $2 * h$ unit of time. □

6.3 Castle-First Strategy

6.3.1 Introduction

In the Castle-First Strategy, all the agents have only local visibility. But the leader agent in each exploration group has the map of the graph in its memory. Furthermore, the leader agents are endowed with the ability of clone.

In the Castle-First Strategy, we build some castles based on the SNR of the nodes. More than one group of agents are sent to explore the graph at the same time. Their exploration of the graph is separated into many "sub-exploration"s and each "sub-exploration" begins where a group of agents are initially residing and ends with a new unexplored castle. The map of the graph with all the castles being pointing out is recorded on the whiteboard on nodes with more than two neighbours (called the *intersections*) so when agents move across these nodes, they can update the information on the whiteboard (for example, changing the state of some castles into explored and update its own memory When agents cannot find an unexplored castle in the graph, they terminate.

6.3.2 Initialization

Based on the graph marked SNR for every node, we built another graph called "Castle Graph". We now give the definition of "Castle":

Definition 1. *(1): A node with a single neighbour is a Castle. (2): A group of connected nodes with the same SNR is a Castle; (3): A node with more than two neighbours with lower SNR than itself, is a Castle.*

Some examples of castles are shown in Fig 6.4

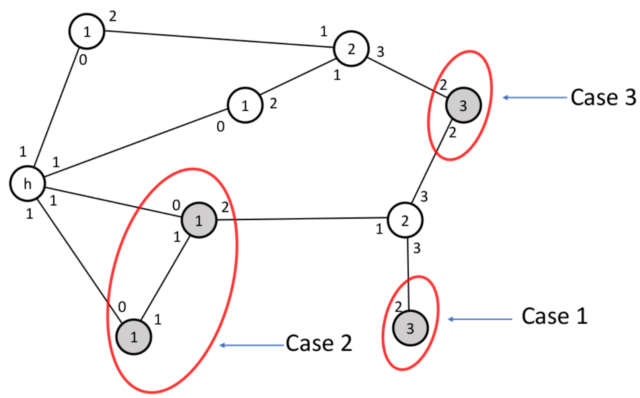


Figure 6.4: Examples of Castles

After the processing, we have a new graph called "Castle Graph", and a "Castle Graph" is a graph with the indication of all the castles positions. This graph is presented on the whiteboard on the nodes which have more than two neighbours, so when the agent reaches this node, it can read the graph to update its own information or using its own information to update the "Castle Graph".

6.3.3 Exploration Phase

The Exploration Phase of the Castle-First Strategy is shown in Figure 6.5 and is introduced in detail in the following.

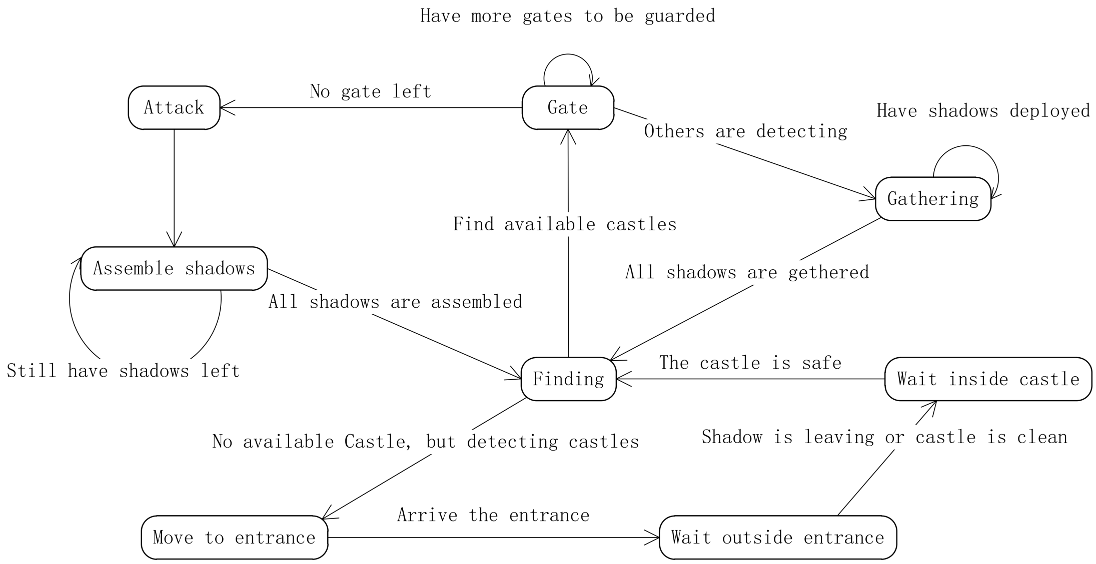


Figure 6.5: Exploration phase of the Exploration Phase

There are two principles in the exploration phase for the agents:

1. When agents move in an unexplored area, they should strictly move from node with smaller SNR to node with bigger SNR and should follow the cautious walk; on the other hand, when they move in an explored area, they can move in any direction (so, also from bigger SNR to smaller SNR) and when they move in this area, they do not do it with cautious walk (for example, when they plan to explore node v from node u, the LA can directly move to node v when node v is in the explored area), which is called "normal walk".

2. The castle $\mathcal{C}$ selected to be the destination of the "sub exploration" should meet the following requirement: assuming that the SNR of $\mathcal{C}$ is a and that the agents start the "sub exploration" from node v, then for every node w with SNR equal to $a-1$ connected to C, there should be a route from v to w and no other unexplored or "under-exploration" castle(s) should exist on these routes. In another word, when the agents move from v to one of these nodes, they do not need to "attack" other

castles.

There are two kinds of agents in this strategy: *Leader Agents* (LA) and *Shadow Agents* (SA) and the LA has the map of the whole graph in its memory. When more agents are needed, the LA can clone the necessary agents. At the beginning of the exploration phase, the LAs are in the homebase, so they each pick a different castle to be the destination of the their first "sub exploration" and since they have the map of the graph, they can compute a shortest route for the "sub exploration" of their castle. Each LA clones as many agents as needed to decontaminate the chosen castle. At any point in time, the agents can be in three states: *Finding-Castle*, *Attacking-Castle* and *Waiting-in-Line*.

Initially, the status of all agents are "Finding-Castle". Consider a group of agents with status of "Finding-Castle", if the LA in that group can find an available castle, then the status of this group changes into "Attacking-Castle". If, during the attack, the agents find the BV, they will proceed to the next phase, otherwise, they will change state into "Finding-Castle" again; if the LA in that agent group cannot find an available castle but there are still castle unexplored in the graph, then the status of this group changes into "Waiting-in-Line".

(1) From "Finding-Castle" to "Attacking-Castle"
On the way to their destination (the castle), the agents move in cautious walk in the unexplored area and normal walk in the explored area: that is, when exploring a node u from node v, one of the SAs moves to node u, when the node u is safe, it returns to node v and moves to node u with the LA and the other SAs; when node u contains a BV, then the LA knows the existence of the BV by receiving the clone of the BV.

When/if one of the agent is destroyed by the BV, then the elimination phase begins.

Along the route to the group's destination, the LA updates the information of the intersections (changing the state of the destination castle to be "under exploration") and also reads information found at the intersections, if it finds that the destination castle of its "sub exploration" has been already explored or is under exploration, then the status of its group changes into "Finding-Castle" again. If not, then this group reaches one of the destination castle's lower SNR neighbour and the LA starts to arrange the SAs to guard all the lower SNR neighbour(s) of the castle.

After the arrangement of the SAs, it should be ensured that all the lower SNR neighbours of the castle are guarded. Consider a castle with m nodes $\{x_1, \ldots, x_m\}$ and let $\{y_1, \ldots, y_h\}$ be the sequence of neighbouring nodes of the castle that should be guarded. That is, the LA would move sequentially from y_1 to y_h. The LA moves sequentially, placing two agents in the last node connected to each x_i in the sequence and one agent in each of the other neighbours so that all these y_i nodes are covered and m of them, one per castle node, contain two agents (the Attacking Agents). For example, in Figure 6.6, let $\{y_1, y_2, \ldots, y_{10}\}$ be the sequence of nodes needing guarding, then after the arrangement, there are two agents in y_2, y_4, y_6, y_8, y_{10} and one agent in the other nodes. When the arrangement of the last agent(s) is completed, the LA informs the Attacking Agents about the exact time when they have to move simultaneously to the castle nodes to permanently destroy the BVs.

When one agent group starts to surround the castle, by which we mean that the LA has placed SAs in at least one lower SNR of the castle, it is possible that another agent group has started to surround the castle as well from a different side (even if the LA updates the information when it moves, still the consistency of all the information cannot be guaranteed). The LA of an agent group realizes this by meeting an SA from a different group when arranging the SAs. We prefer to avoid conflict when two agent group explore one castle (SAs from two different agent group move to the castle nodes). The LA can compute the time when the arrangement is finished and record it on a timestamp, so when the LA places the SAs, it leaves timestamp with these SAs. By doing this, every SA from one agent group guarding the castle hold a timestamp recording when does the arrangement end and if a LA meet a SA from other agent group with a earlier timestamp, it moves back to collect the SAs in its group that have been placed and the status of this agent group turn into “Finding-Castle”. If the LAs have the same timestamp, then the conflict cannot be avoided, but this case barely happens.

When the arrangement is done, the LA resides in the last node needing guarding and in the next unit of time, all the Attacking Agents move to the castle nodes. Note that at this time, if there is a BV in the castle, not all the guarding agents can receive a BV clone. For example, in Fig6.6, when the BV is triggered only the guarding agents 1 and 2 receive the clones.

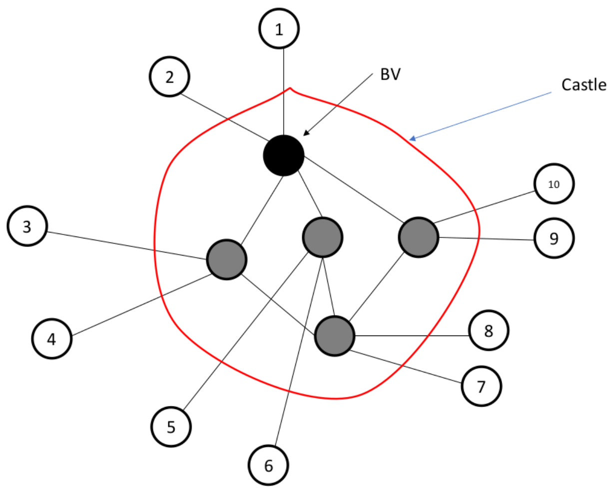
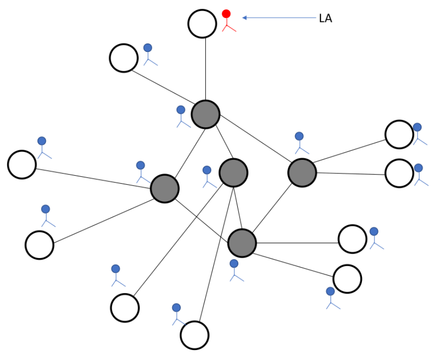


Figure 6.6: An example shows that if there is a BV in the castle, in some cases not all the guarding agents can receive a BV clone

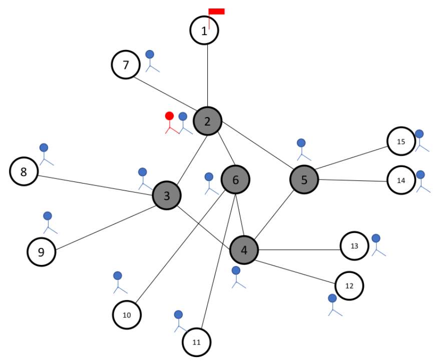
So after the “attacking” by the “Attacking agent”, the LA should execute a “Double Patrol” to check if there was a BV in this castle and to locate it if there was. We now introduce the idea of “Double Patrol”.

1. The LA first computes a route to traverse all the castle nodes, (say the route is $x_0 \to x_1 \to \ldots, x_m$ assuming that there are m nodes in the castle). In this first “patrol” (see Figure 6.7), the LA first leaves a flag saying: “First Patrol” and then moves to one castle node, it tells the SA residing in that castle node to collect the SAs residing in all its lower SNR neighbours and finally moves to that castle node

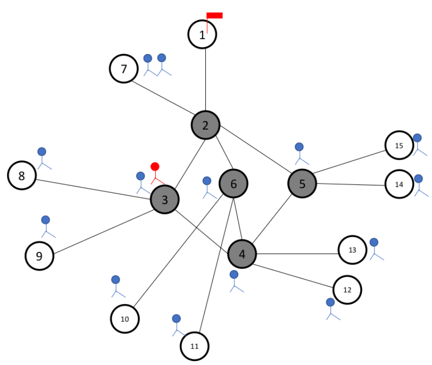
with all these collected SAs. Also, when the SA collects the agents residing in the lower SNR neighbours, it places a flag on that neighbour node saying: "First Patrol". After the LA traverses all the castle nodes, it can easily know whether there is a BV in the castle: if there is no BV in the castle, then there should be one agent residing in every castle node and the LA would start the "second patrol". Otherwise, there is a BV and its clones have moved to higher SNR neighbours of this node. If the LA realizes that there is a BV, it stops traversing the remaining castle nodes.



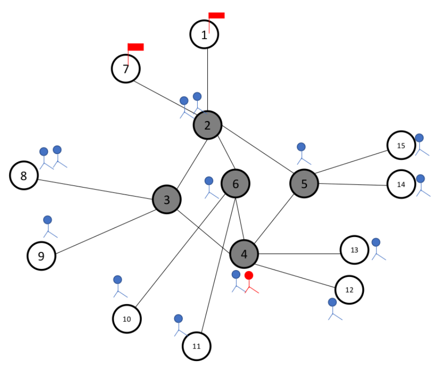
(a) T_a



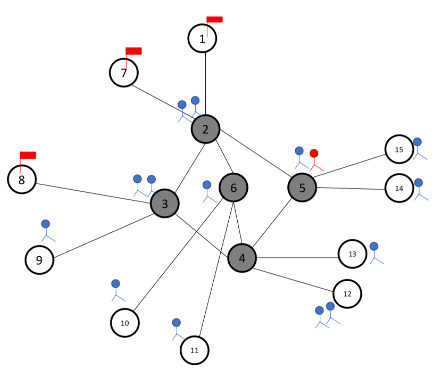
(b) T_{a+1}



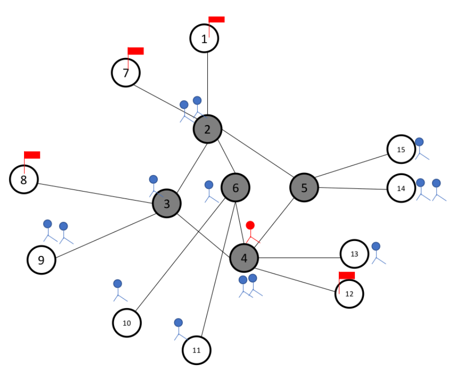
(c) T_{a+2}



(d) T_{a+3}

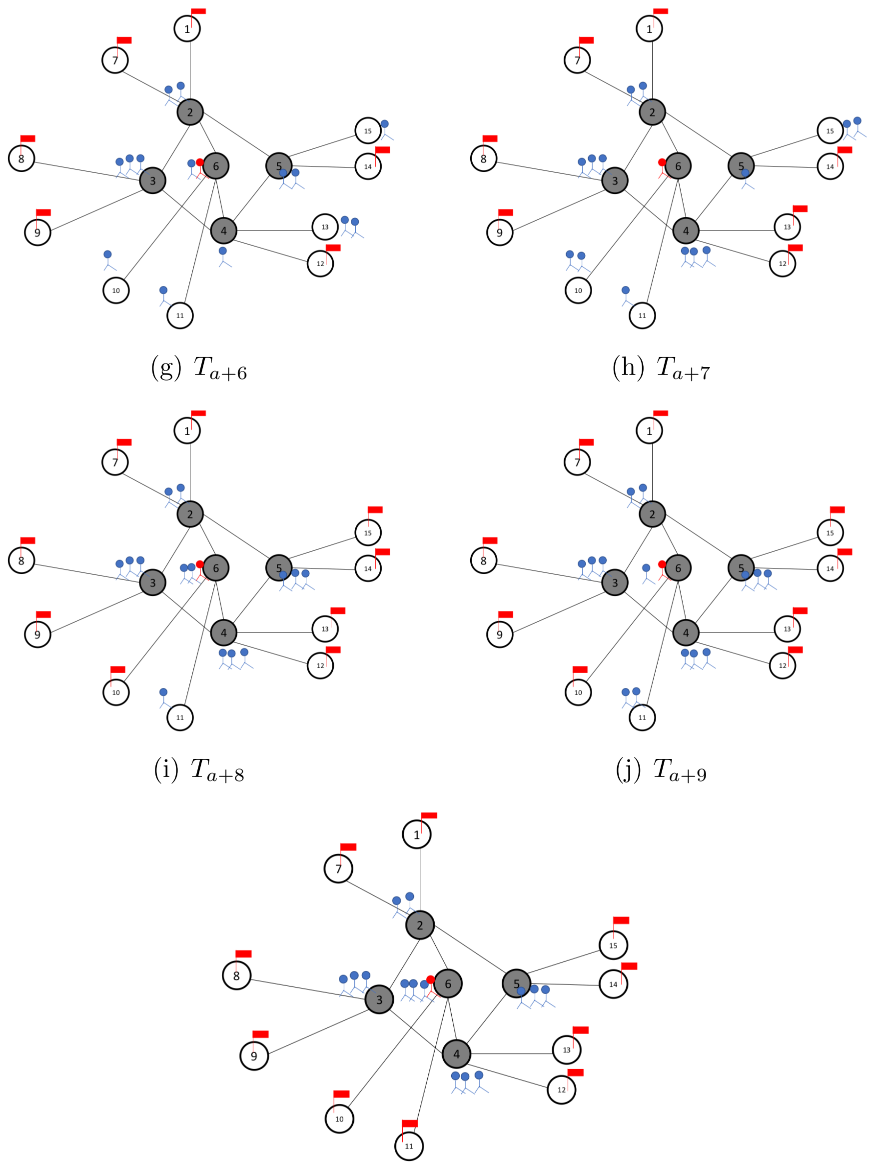


(e) T_{a+4}



(f) T_{a+5}

Figure 6.6: An example of "FirstPatrol"



(g) T_{a+6} (h) T_{a+7}

(i) T_{a+8} (j) T_{a+9}

(k) T_{a+10}

Figure 6.7: An example of “FirstPatrol”

2. After the first patrol, if there is a BV the Elimination phase starts; otherwise, the “Second Patrol” starts. In the “Second Patrol” the LA moves along the reverse route of the “First Patrol” which is $x_m \rightarrow x_{m-1} \rightarrow \ldots, x_0$. While doing so, it collects all the SAs in the castle node and it updates the information on the whiteboards on the castle nodes saying that the castle is explored.

After the “Second Patrol”, the status of this agent group changes into “Finding-Castle” again.

(2) From “Finding-Castle” to “Waiting-in-Line”

When the status of the agent group changes into “Waiting-in-Line”, then it means that the LA in that agent group cannot find an available castle but there are still castles unexplored in the graph. At this time, the LA chooses a castle marked “under exploration” and moves there with its agent group. When the group arrives in one of the lower SNR neighbours of that castle (the “Guarding Node”) three situations may happen:

1. Case 1: The agent group arrives at the Guarding node finding that no SA(s) and no flag in that Guarding Node. In this situation, the agent group who is expected to attack this castle may not have yet arrived to this castle or may not have finished the arrangement of the SAs. Then the agent group should wait at that guarding node until there is an SA and follow the instruction of Case 2 when this happens.

2. Case 2: The agent group arrives at the Guarding node finding at least one SA. In this situation, it means that the agent group who is expected to attack the castle has finished arranging the SAs but has not finished the “First Patrol”. So, this agent group stays with the SA until the SA is called by another SA and moves into the castle. After moving into the castle node, the agent group should wait for the LA of the agent group that is attacking the castle, to move to this castle node again because at this time the LA would update the situation in the castle: whether there is a BV or not. If there is a BV in the castle, then the agent group that is waiting in the castle node ends its exploration phase and start the elimination phase; otherwise, the status of this agent group changes into “Finding-Castle”.

3. Case 3:The agent group arrives at the Guarding node and finds that there is a flag. This means that the LA of the agent group who is excepted to attack the castle is doing (or has finished) the “First Patrol”, or it is doing the “Second Patrol”. In

this situation, the agent group moves into the castle node which is connected to the guarding node and wait for the LA of the attacking group to move to this castle node again. As in Case 2, this agent group knows if there is a BV in the castle: if there is a BV, then the agent group starts the elimination phase, otherwise its status changes into “Finding-Castle”.

6.3.4 Elimination Phase

When the BV is triggered, the location of the BVs can be in a castle or outside the castles. In both situation, the lower SNR neighbours of the BV node or the castle containing the BV have been guarded by the agents. In other words, when the BV is triggered, only the clones spreading to the higher SNR neighbours survive and their locations are exposed. Also, the locations of all the neighbours of these new formed BVs are exposed.

So, the LA computes the routes from where it resides to all the nodes needing guarding, including both the lower and the higher SNR neighbours (Surrounding Nodes). If the location of the original BV is outside the castle, then all the SAs stay with the LA, so the LA simply sends the SAs to these Surrounding Nodes. If the original BV is in the castle, then after the “Double Patrol”, the LA knows the location of the original BV, so it computes the routes from where it resides to all the Surrounding Nodes and send the SAs to them (note that all the SAs are with the LA after the “Double Patrol”). Also note that since the graph is biconnected, there is always a route to reach the surrounding node bypassing the new formed BV nodes. Assuming that there are k higher SNR neighbours of the original BV, then the LA sends another k agents to the new formed BV nodes and permanently destroy the BVs.

Chapter 7

Experimental Study on BVD in Arbitrary graph

In this Chapter, we experimentally investigate the problem of black virus decontamination by studying the Castle-First Strategy described in Chapter 6 and comparing it with the Greedy Exploration Algorithm [5]. The simulator we built is called ArbiBV. We introduce its modeling methods and execution. The behavior, properties, and performance have been investigated through extensive computer simulation runs. The simulation results confirm the expected behavior/properties of the solution protocol.

7.1 The simulator

7.1.1 Simulation Model

Our simulator only execute the exploration phase, since the parameters (for example, the number of agents, the execution time ...) can be easily calculated after the location of the BV is detected (the time for the elimination phase is $O(1)$). In the simulator, we consider the worst possible scenario in terms of BV position, which actually corresponds to having no BV in the system, that is, the agents have to explore the whole graph. The environment where the agents operate is a network modeled as a simple arbitrary undirected connected graph where each node has a unique ID (without loss of generality, from 1 to n where n is the number of node in the graph). An agent is modeled as an entity with computational or information processing capability. All the agents follow the same protocol while their roles and the states can be different (for example, some of the agents are Leader Agents (LA) while some of the agents are Shadowing Agents (SA)). Note that our simulator does not have a Graphical User Interface(GUI). For monitoring purposes, it provides a textual description of the agents' activities only. In each run of the simulator, we provide the number of nodes, the average degree of the graph, and the simulator generates an arbitrary graph based on that. After each run, the simulator calculates the number of agent deployed and the units of time elapsed.

7.1.2 Simulation Sample graph

In this section, we compare the number of agents and the execution time between the CASTLE-FIRST Strategy and the sequential strategy of [5], also between CASTLE-FIRST Strategy with different settings (sending different numbers of groups to explore the graph). When we compare our strategy with the sequential strategy. We run our protocol in different graph settings; more precisely, network *size* n, network *connectivity density* NC, and number of groups of agents deployed m, where the connectivity density is defined as the ratio of the number of links of the graph with the number of links in a complete graph with the same number of nodes. In our experiments we consider: $n = 20, 40, 60, 80, 100$, $NC = 10\%, NC = 20\%, \ldots, NC = 100\%$, $m = 1, 2, 3, 4$.

indicate what values of density are considered between $20 and 100\%$

For each specific setting (size of the graph, network connectivity density, number of groups of agents), we run the experiment for 50 times: the average execution time and the average number of agents needed would be the result of the execution time and the number of agents of this setting. So overall, we run 9020 experiments to obtain statistic data for our strategy.

By doing the comparison, we are interested in observing the time complexity of our strategy compared to the sequential one, and the cost (in terms of additional agents) that we must pay for the faster speed.

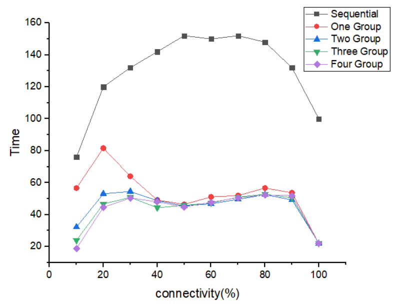
When we compare the results of different strategy settings of Castle-First Strategy, we are interested in the following aspects:

1. What is the relationship between the execution time and the number of agent groups that participate in the strategy;
2. What is the impact of various topological factors (size, connectivity, ...) on the efficiency of the strategy.

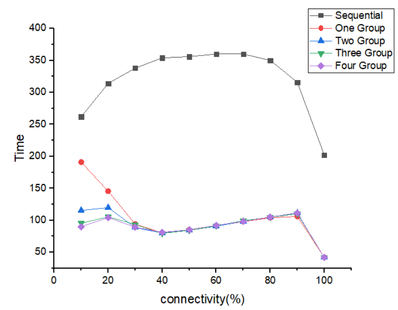
7.2 Simulation Results

In this section, we compare the Castle-First Strategy with the Sequential Strategy (Greedy Exploration). As a worst case scenario, we let the exploration proceed until all nodes are explored, without placing any black virus. For each network size, we create 10 connectivity levels and run our protocol with the number of groups ranging from one to four.

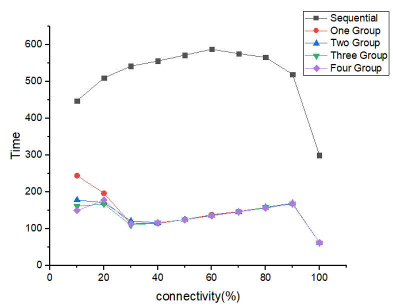
Simulation results show that the Castle-First Strategy is never slower than the Greedy Exploration, even when only one group of agents explores the graph, and the extra cost in terms of the number of agents that we pay is acceptable. The time needed to finish the exploration of the graph in Greedy Exploration and in the Castle-First Strategy (where the number of exploring group ranging from one to four) is as shown in Figure 7.1



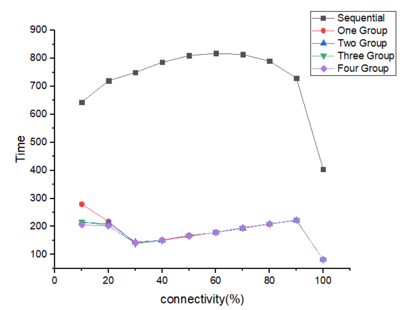
(a) size of graph:20 nodes



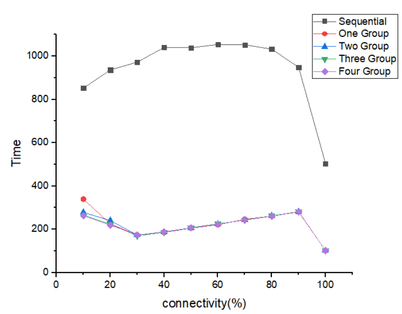
(b) size of graph:40 nodes



(c) size of graph:60 nodes



(d) size of graph:80 nodes



(e) size of graph:100 nodes

Figure 7.1: Comparison between GREEDY Exploration and CASTLE-FIRST Strategy

From Figure.7.1, we have the following observations about the time complexity:

1. <u>Sequential Strategy (GREEDY Exploration) vs CASTLE-FIRST Strategy:</u>

   In all sizes of the graph, the time of the GREEDY Exploration increases gradually, reaching a maximum at 40%-60% connectivities, after that, it gradually decreases. It should be pointed out that the time costs are close for different graphs at 100% connectivity level and are comparable or less than those at all 10% connectivity level. While in the Castle-First Strategy, the time cost first increases slightly and then decreases gradually to a minimum at 28%-40% connectivities, after that, it gradually increases to values comparable to those at all 10%-20% connectivities, and then decrease sharply to reach a minimum when the connectivity is 100%. Note that, though experiencing an increase from connectivity of 10 to 20 when the size of node is 20, the time increases less and less obviously as the size of the graph grows (an exception is the case of "one group" of agents, for all sizes, where the time cost always decreases until it reaches a local minimum). When the size of the graph is 80, the time cost of the Castle-First Strategy executed by more than one group stays at almost the same level from connectivity 10% to 20%. When the size of the graph is 100, the time directly decreases to the local minimum. This behavior is reasonable because, as the size of the graph grows, even maintaining the same connectivity, the average degrees of nodes in the graph increase, which means the number of routes from one node to another increase to some extent, so when the agents move from one castle to another, they might have some shorter routes to choose, which saves time. The behavior of the "one group" agent is largely influenced by this factor, so, except in the graph of 20 nodes, the time cost with a single group does not experience a slight increase but decreases directly to the local minimum.

After reaching the local minimum, the movement starts to increase. The reason is that, although there are shorter routes to choose as the connectivity increases when moving to the next castle (which saves time), this advantage becomes less obvious because the length of the route from one node to another does not change significantly as the connectivity increases. At the same time, as the graph becomes denser, the castles are becoming larger, so we need to complete the "Double Patrol" in large castles, which costs a significant amount of time. The result of the counteracting is that the movement increases after the local minimum movement.

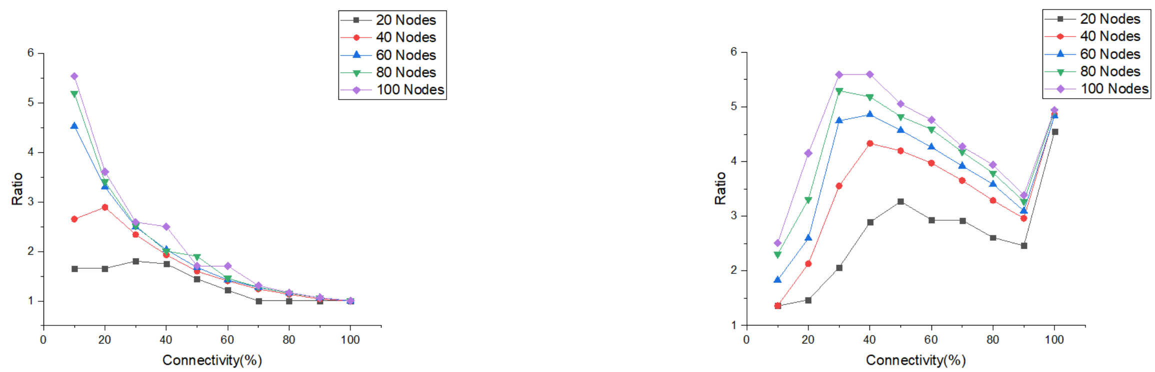
When the connectivity is larger than 90%, we observe the formation of a single very large castle. As a consequence, in the exploration phase, the agents have to explore one castle, which results in a decrease of movement.

2. Castle-First Strategy varying the number of groups of agents.

   From Figure 7.1, we can see that when the connectivity is larger than 30%, the movements are almost the same no matter how many groups are sent to explore the graph. This can be explained; in fact, according to our algorithm, some of the castles in a graph have to be explored strictly sequentially (this is the case, for example, of some castles that are not accessible without first exploring others). This means that, in some cases, even if there are castles unexplored and groups of agents available, full parallelism cannot be exploited. In these situations, sending more groups to explore the graph does not decrease time. So we prefer to use one group of agents executing the Castle-First Strategy when the connectivity is larger than 20%.

   Another important observation is that as the size of the graph grows, the connectivity from where different group members perform similarly or almost the same is becoming smaller.

Based on the data presented in Figure 7.1, we now compare the time and the number of agents used by these two strategies. Since the time cost between the Castle-First Strategy executed by different groups of agents is similar from connectivity 10% to 100% (especially when the connectivity is larger than 20%), we only compare the statistics between the Greedy Exploration and the the Castle-First Strategy with one group of agent. In figure 7.2, we show the ratio between the number of agents employed in the Castle-First Strategy using a single group and the number of agents used in the Greedy Exploration in the same situation. We also show the ratio between the time incurred by the Castle-First Strategy using one group of agents and the time incurred by the Greedy Exploration.



(a) Ratio between the number of agents used in Castle-First with a single group and the number of agents used in Greedy

(b) Ratio between time incurred by Castle-First using a single group of agents and time incurred by Greedy

Figure 7.2: Ratio between time and agent cost of Castle-First using a single group and Greedy in the same situation.

In Figure 7.2 (a) the ratio is equal to number of agent used in the Castle-First Strategy using a single group divided by the number of agents used in Greedy Exploration (for convenience, we call it ratio 1); in Figure 7.2 (b), the ratio is equal to the time cost incurred by the Castle-First Strategy using a single group divided by the time cost incurred by the Greedy Exploration (for convenience, we call it ratio 2).

From the Figure 7.2, we can make the following observations:

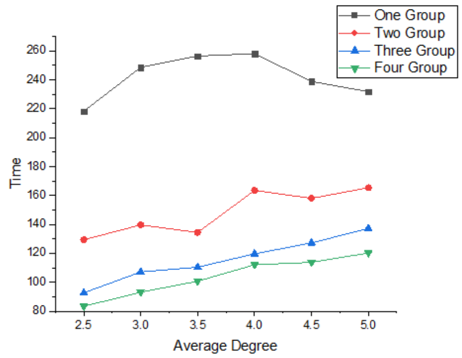
1. As the connectivity grows, we can see that ratio 1 is decreasing, and when the connectivity is larger than 30%, the ratio does not change a lot. When the connectivity is larger than 70%, the ratio is near 1. This tells us that when the connectivity is smaller than 20%, we employ a much larger number of agents (up to 5.5 times) than in the Greedy Exploration, while when the connectivity is between 30% and 60%, we use 1.2 to 2.5 times as many agents as that in the Greedy Exploration. When the connectivity exceeds 70%, we use almost the same number of agents as the ones used in the Greedy Exploration.

2. Ratio 2 first increases reaching the local maximum when the connectivity reaches 30% to 50%, then decreases to the local minimum when the connectivity reaches

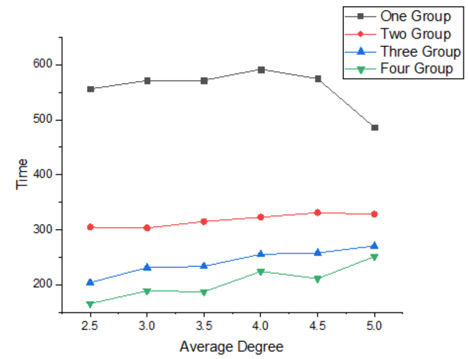
90%; finally, it increases sharply to reach a maximum when the connectivity reaches 100%.

3. Based on the two observations above, we come to the conclusion that, when the connectivity of the graph reaches 30% to 50%, the Castle-First Strategy using a single group of agents works most efficiently.

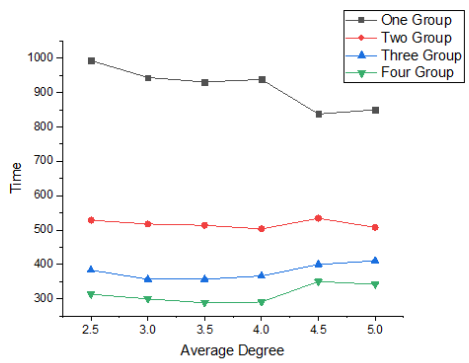
From the data we have shown in Figure 7.1 it appears that, except for the case of a single group, the performances of different number of groups are similar, even when the connectivity is only 10%, so we compute the time cost at intervals of 10% connectivity. To provide more precise statistics describing the behavior between 0 and 10% connectivity (the connectivity from which the performance of different group numbers is similar or almost the same) we should look in more details at this interval. It is obvious that for different graph sizes, even with the same connectivity, the degrees of the nodes are different so we will use the "Average Degree" to see if the results of the Castle-First Strategy executed by different number of groups of agents have large difference depending on the average degrees of their nodes. The average degree of a graph is an important factor because it influences how the castles of the graph form. From the conclusion of Figure 7.1, we know that when the connectivity is larger than 10% to 20%, there are no big differences between the time incurred by different groups of agents, so now we would like to start from a small degree of 2.5 and increase by 0.5 at each time (as shown in Figure 7.3)



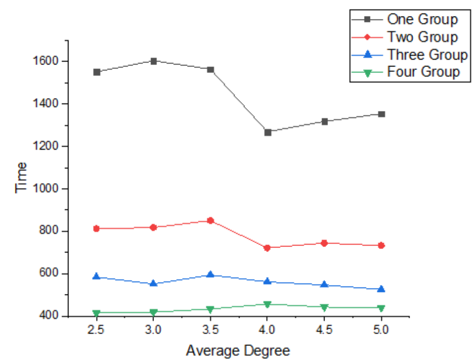
(a) size of graph:50 nodes



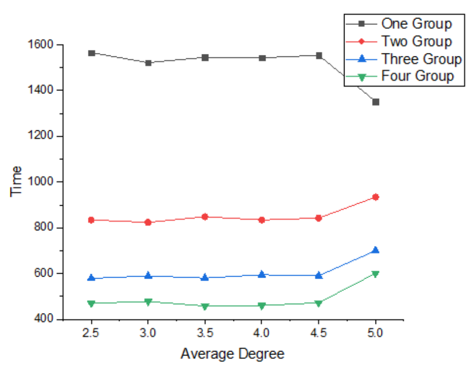
(b) size of graph:100 nodes



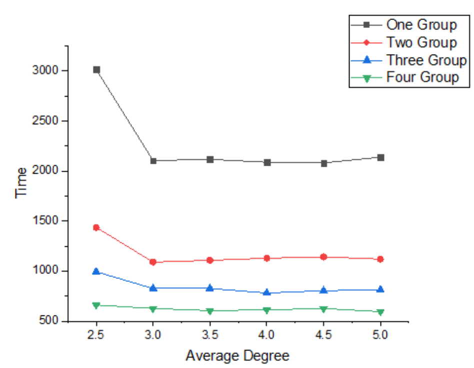
(c) size of graph:150 nodes



(d) size of graph:200 nodes



(e) size of graph:250 nodes



(f) size of graph:300 nodes

Figure 7.3: Time cost by different number of group

From the data obtained, we can make the following observations:

1. For different group sizes, the largest difference in time cost can be observed between the strategy executed by one group and two groups of agents (although there are still some decreases when we increase the number of groups executing the strategy, they

are much smaller). Also, the difference in time cost when employing different group numbers is smaller as the average degree grows.

2. Excluding now the case of a single group, let us focus on the performance of other group numbers. When the size of the graph is 50, we can see that the performances of different group numbers are similar at the beginning (the difference between them is smaller than 50). When the size increases to 100, the difference between different group members is more evident, and appears at the average degree of 2.5. What should be pointed out is that such a big difference exists between the time cost by two groups and three groups of agents. So, if we want to execute the strategy and decrease time when the size of the graph is smaller than 100 nodes, at most two groups of agents are recommended.

   We also run experiments on larger graphs (when the number of nodes is bigger than 100). First note that, according to our conclusion based on Figure 7.1, as the size of the graph grows, the connectivities from where different group members perform similarly become smaller. As a result, the average degree from where different group members perform similarly increases, but very slowly. That is, like in the case of small size graphs, the performance of different number of groups in larger graphs is still similar when the average degree is larger than a certain number (this number might increase as the size of the graph grows, but the increase is slow). From Figure 7.1 (c) to (f), we can see that there are large differences between time cost by different group numbers (even between the time cost by three groups or even four groups of agent). When the size of the graph is 250 or 300, we can see that there are still big differences between the performance of three groups and four groups of agents at an average degree of 5.0 (which is the maximum average degree shown in Figure 7.3) We can see that at this interval (average degree from 2.5 to 5.0), as the size of the graph grows, the advantage of using more groups of agents is becoming more obvious, and this interval is longer as the size of the graph becoming larger.

Chapter 8

Concluding Remarks and Future Work

8.1 Summary

Mobile agents are widely used in distributed and network systems, but their employ can cause several security issues. Particularly, a malicious agent can cause computer nodes to malfunction or crash by contaminating or infecting them (*harmful agent*); a contaminated or infected host can in turn destroy working agents for various malicious purposes (*harmful host*). A problem triggered by the harmful agent is called $Intruder\,Capture$(IC) which focuses on deploying a group of mobile agents to capture an extraneous mobile agent (the intruder) that moves arbitrarily fast through the network and infects the visiting sites. A problem triggered by the presence of harmful hosts (also called $Black\,Holes$) is called $Black\,Hole\,Search(BHS)$, the focus of which is to locate the positions of these $Black\,Holes$ in the topology. These two problems have been widely studied in many variants.

The BVD problem, integrating in its definition both the harmful aspects of the classical

BHS problem with the mobility of the classical IC problem, was first introduced by [5]. The main focus of [1, 5, 6] add other references on BV was on the design of sequential protocols for agents to complete this task causing minimum network damage and sacrifying the minimum number of agents.

In this thesis, we propose parallel strategies to deal with the BVD problem, focusing on employing more agents to explore the graph in parallel, thus decreasing the time needed to finish the decontamination.

This chapter highlights the main contributions of this thesis to the parallel BVD problem:

The first part of the thesis described the terminology and the model. In this thesis, we only consider the situation of $Fertile$ BV. In this case, we need to protect the network from further spread after detecting the original BV. A basic principle that monotonicity is necessary for optimality should be obeyed: in every network, to minimize the spread, a solution protocol must be monotone.

The second part of this thesis focused on the parallel BVD problem for three important classes of interconnection network topologies:(multi-dimensional) grids, tori and chordal rings. For each topology, we provided a parallel strategy and analyzed the number of agents, the time cost, the number of movements and the casualties. Comparisons between the parallel strategy and the sequential strategy in each topology are also made showing that the better performances in terms of time are accompanying by a small increase in the number of agents employed.

The third part of this thesis focused on the parallel BVD problem in an arbitrary graph. We proposed two strategies to deal with the problem: *Flood Strategy* and CASTLE-FIRST Strategy. The first strategy explores of the graphin optimal time (proportional to its diameter), but it employs many more agents than the sequential strategy; on the other hand, the CASTLE-FIRST Strategy completes the exploration of the graph improving significantly the time complexity with respect to the sequential strategy, but using a much lower number of agents than the ones used in the *Flood Strategy*. In order to achieve this goal, we had to do some preprocessing work: namely, computing the length of the shortest route from the homebase to each node; using whiteboards to record some useful information.

The last part of this thesis shifts the theoretical study on an arbitrary graph (the CASTLE-FIRST Strategy) to experimental investigation. A large number of simulations on different sizes of graphs with many connectivity densities are carried out. The experiment results show that even using one group of agents to explore the graph, the time cost by the CASTLE-FIRST Strategy is much lower than the sequential strategy.

Additionally, as we employ more agents than in the sequential strategy, when the connectivity of the graph reaches 30% to 50% the CASTLE-FIRST Strategy using a single group of agents works most efficiently. That is, the ratio between the time incurred by CASTLE-FIRST using a single group of agents and the time incurred by the sequential (GREEDY)one is much larger than the ratio between the number of agents used in CASTLE-FIRST with a single group and the number of agents used in GREEDY. When the connectivity is low(less than 10%), as the size of the graph grows, the advantage of using more groups of agents is becoming more evident. That is, when we use more groups of agents to explore the graph, the time cost still experiences a great decrease.

8.2 Open Problems and Future Research

The results of this thesis open many research problems and pose new questions.

1. In addition to grids, tori, and choral rings, future research can be extended to the design of parallel strategies for BVD in other network topologies, like the hypercube for example.

2. In the parallel strategies that we propose to deal with the BVD problem in the arbitrary graph, we have to do some preprocessing work (shortest path computation) and we require full knowledge of the topology, future research can be extended to deal the problem when less knowledge of the graph is available.

3. We investigate the problem with a single BV, future research can be extended to study the problem with more than one BV.

References

[1] M. Alotaibi. Black virus disinfection in chrodal rings. Master's thesis, University of Ottawa, 2014.

[2] L. Barrière, P. Flocchini, F. V. Fomin, P. Fraigniaud, N. Nisse, N. Santoro, and D.M. Thilikos. Connected graph searching. *Information and Computation*, pages 1–16, 2012.

[3] L. Barrière, P. Flocchini, P. Fraignaud, and N. Santoro. Capture of an intruder by mobile agents. In *14th Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 200–209, 2002.

[4] L. Blin, P. Fraignaud, N. Nisse, and S. Vial. Distributed chasing of network intruders. *Theoretical Computer Science*, 2006.

[5] J. Cai, P. Flocchini, and N. Santoro. Decontaminating a network from a black virus. *International Journal of Networking and Coputing*, pages 151–173, 2014.

[6] J. Cai, P. Flocchini, and N. Santoro. Distributed black virus decontamination and rooted acyclic orientations. In *CIT/IUCC/DASC/PICom*. IEEE, 2015.

[7] J. Chalopin, S. Das, A. Labourel, and E. Markou. Black hole search with finite automata scattered in a synchronous torus. In *25th International Symposium on Distributed Computing(DISC)*, pages 432–446, 2011.

[8] J. Chalopin, S. Das, A. Labourel, and E. Markou. Tight bounds for black hole search with scattered agents in synchronous rings. *Theoretical Computer Science*, pages 70–85, 2013.

[9] J. Chalopin, S. Das, and N. Santoro. Rendezvous of mobile agents in unknown graphs with faulty links. In *21st International Symposium on Distributed Com- puting (DISC)*, pages 108–122, 2007.

[10] C. Cooper, R. Klasing, and T. Radzik. Searching for black-hole faults in a network using multiple agents. In *10th International Conference on Principles of Distributed Systems (OPODIS)*, pages 320–332, 2006.

[11] C. Cooper, R. Klasing, and T. Radzik. Locating and repairing faults in a network with mobile agents. *Theoretical Computer Science*, pages 1638–1647, 2010.

[12] J. Czyzowicz, S. Dobrev, R. Královič, S. Miklík, and D. Pardubská. Black hole search in directed graphs. In *16th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 182–194, 2009.

[13] J. Czyzowicz, D. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in synchronous tree networks. *Combinatorics, Probability and Computing*, pages 595–619, 2007.

[14] J. Czyzowicz, D. R. Kowalski, E. Markou, and A. Pelc. Complexity of searching for a black hole. *Fundamenta Informaticae*, pages 229–242, 2006.

[15] J. Czyzowicz, D.R. Kowalski, E. Markou, and A. Pelc. Searching for a black hole in tree networks. In *In Proceedings of the 8th International Conference on Principles of Distributed Systems*, pages 67–80, 2004.

[16] S. Das, P. Flocchini, and N. Santoro. Mobile search for a black hole in an anonumous ring. *Algorithmica*, pages 67–90, 2007.

[17] D. Dereniowski. Connected searching of weighted trees. *Theoretical Computer Science*, pages 5700–5713, 2011.

[18] S. Dobrev, P. Flocchini, R. Královič, and N. Santoro. Exploring an unknown dangerous graph using tokens. *Theoretical Computer Science*, pages 28–45, 2013.

[19] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in arbitary networks:optimal mobile agents protocols. *Distributed Computing*, pages 1–18, 2006.

[20] S. Dobrev, N. Santoro, and W. Shi. Using scattered mobile agents to locate a black hole in a unoriented ring with tokens. *Int. Journal of Foundations of Computer Science*, 2008.

[21] P. Flocchini. Contamination and decontamination in majority-based systems. *Journal of Cellular Automata*, pages 183–200, 2009.

[22] P. Flocchini, M. Huang, and F. Luccio. Decontaminating chordal rings and tori using mobile agents. *International Journal of Foundations of Computer Science*, pages 547–563, 2007.

[23] P. Flocchini, M. Huang, and F. Luccio. Decontamination of hypercubes by mobile agents. *Networks*, pages 167–178, 2008.

[24] P. Flocchini, M. Kellett, P. Mason, and N. Santoro. Map construction and exploration by mobile agents scattered in a dangerous network. In *24th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–10, 2009.

[25] P. Flocchini, F.L. Luccio, and L.X. Song. Size optimal strategies for capturing an intruder in mesh networks. *International Conference on Communications in Computing (CIC)*, pages 200–206, 2005.

[26] P. Flocchini and N. Santoro. *Distributed Security Algorithms for Mobile Agents*. Mobile Agents in Networking and Distributed Computing, 2012.

[27] F.V. Fomin and D.M. Thilikos. An annotated bibliography on guarabteed graph searching. *Theoretical computer science*, pages 236–245, 2008.

[28] F.V. Fomin and D.M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, pages 236–245, 2008.

[29] P. Glaus. Locating a black hole without the knowledge of incoming link. In *5th Int. Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSOR)*, pages 128–138, 2009.

[30] F. Hohl. Time limited black box security: protecting mobile agents from ma- licious hosts. In *Mobile Agents and Security, Lecture Notes in Computer Science*, pages 92–113, 1998.

[31] F. Hohl. A framework to protect mobile agents by using reference states. In *20th Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 410–417, 2000.

[32] D. Ilcinkas, N. Nisse, and D. Soguet. The cost of monotonicity in distributed graph searching. *Distributed Computing*, pages 117–127, 2009.

[33] N. Imani, H. Sarbazi-Azadb, and A.Y. Zomaya. Capturing an intruder in product networks. *Journal of Parallel and Distributed Computing*, pages 1018– 1028, 2007.

[34] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Hardness and approxima- tion results for black hole search in arbitrary networks. *Theoretical Computer Science*, pages 201–221, 2007.

[35] R. Klasing, E. Markou, T. Radzik, and F. Sarracco. Approximation bounds for black hole search problems. *Networks*, pages 216–226, 2008.

[36] A. Kosowski, A. Navarra, and C. M. Pinotti. Synchronous black hole search in directed graphs. *Theoretical Computer Science*, pages 5752–5759, 2011.

[37] F. Luccio, L. Pagli, and N. Santoro. Network decontamination in presence of local immunity. *International Journal of Foundations of Computer Science*, pages 457–474, 2007.

[38] M.G. Rubinstein and O.C.M.B. Duarte. Evaluating the performance of mo- bile agents in network management. In *Global Telecommunications Conference (GLOBECOM)*, number 386-390, 1999.

[39] T. Sander and C.F. Tschudin. Protecting mobile agents against malicious hosts. In *Mobile Agents and Security, Lecture Notes in Computer Science*, pages 44–60, 1998.

[40] P. Shareghi, H. Sarbazi-Azad, and N. Imani. Capturing an intruder in the pyramid. *International Computer Science Symposium in Russia (CSR)*, pages 580–590, 2006.

[41] W. Shi. Black hole search with tokens in interconnected networks. In *11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 670–682, 2009.